Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Основы алгоритмизации и программирования (ОАиП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту на тему

Программное средство «Построения

генеалогического древа»

БГУИР КП I–40 01 01 309 ПЗ

Выполнил

студенn гр. 251003 Жук Я. С.

Проверил: Фадеева Е.П.

Минск 2023

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ

Заведующий кафедрой ПОИТ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

Лапицкая Н.В. 2023г.

ЗАДАНИЕ

по курсовому проектированию

Студенту Жук Яне Сергеевне\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Тема работы Программное средство «Построения генеалогического древа»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2. Срок сдачи законченной работы *03.06.2023г.*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3. Исходные данные к работе *Среда программирования Delphi. Возможность ввода данных с клавиатуры или чтением из типизированного файла. Предусмотрение контроля корректности ввода данных, а именно: текстовые и числовые поля. Предоставление информации о программе из текстового файла. Возможность сохранения данных в типизированный файл. Реализована динамическая структура данных: однонаправленный список. Реализация функций работы с персонами: добавление и удаление персон, изменение их информации, поиск персон, установка родственных связей, графическое представление генеалогического древа и его экспорт.*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке)

*Введение*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*1 Анализ литературных источников и формирование функциональных требований к разрабатываемому программному средству\_*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*2 Проектирование и разработка программного средства* \_\_\_\_\_\_*\_\_\_\_\_\_\_\_\_\_\_*

*3 Тестирование и проверка работоспособности программного средства\_\_\_*

*4 Руководство по установке и использованию программного средства\_\_\_\_\_*

*Заключение \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Список использованных источников\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Приложения* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)

*Схема алгоритма в формате А1*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

6. Консультант по курсовой работе *Фадеева Е.П.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

7.Дата выдачи задания *16.02.2023г.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и процентом от общего обьема работы):

*Раздел 1, Введение к 28.02.2023г. – 10 % готовности работы;\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Раздел 2 к 15.03.2023г. – 30% готовности работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Раздел 3 к 15.04.2023г. – 60% готовности работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Раздел 4, Заключение,Приложения к 20.05.2023г. – 90% готовности работы;*

*оформление пояснительной записки и графического материала к 31.05.2023г. – 100% готовности работы.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Защита курсового проекта с 01.06.2023г. по 03.06.2023г.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

РУКОВОДИТЕЛЬ *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Фадеева Е.П.*

*(подпись)*

Задание принял к исполнению  *Жук Я. С. 16.02.2023г.*

*(дата и подпись студента)*

Содержание

[Введение 6](#_Toc135979224)

[1 Анализ литературных источников 7](#_Toc135979225)

[1.1 Анализ существующих аналогов 7](#_Toc135979226)

[1.1.1 Программное средство «MyHeritage» 7](#_Toc135979227)

[1.1.2 Программное средство «Древо жизни» 11](#_Toc135979228)

[1.1.3 Итоги анализа существующих аналагов 13](#_Toc135979229)

[1.2 Анализ средств разработки программного средства 14](#_Toc135979230)

[1.2.1 Запись 14](#_Toc135979231)

[1.2.2 Деревья 14](#_Toc135979232)

[1.2.3 Линейные однонаправленные списки 15](#_Toc135979233)

[1.2.4 Файлы 15](#_Toc135979234)

[1.2.5 Сортировка с помощью разделения 17](#_Toc135979235)

[1.3 Спецификация функциональных требований 18](#_Toc135979236)

[2 Разработка и проектирование программного средства 20](#_Toc135979237)

[2.1 Описание алгоритмов решения задачи 20](#_Toc135979238)

[2.2 Структура данных 25](#_Toc135979239)

[2.2.1 Структура типов основной программы 25](#_Toc135979240)

[2.2.2 Структура данных основной программы 27](#_Toc135979241)

[2.2.3 Структура данных алгоритма AmountOfBlocks 27](#_Toc135979242)

[2.2.4 Структура данных алгоритма ShowInfoInCard 28](#_Toc135979243)

[2.2.5 Структура данных алгоритма AddToTheList 28](#_Toc135979244)

[2.2.6 Структура данных алгоритма AddToTheCombos 28](#_Toc135979245)

[2.2.7 Структура данных алгоритма GetFullName 28](#_Toc135979246)

[2.2.8 Структура данных алгоритма GoingThroughTree 29](#_Toc135979247)

[2.2.9 Структура данных алгоритма RenamePhotoPath 29](#_Toc135979248)

[2.2.10 Структура данных алгоритма TFwork.SBPersonsFind 29](#_Toc135979249)

[2.2.11 Структура данных алгоритма RemovePerson 30](#_Toc135979250)

[2.2.12 Структура данных алгоритма ActDrawTreeExecute 30](#_Toc135979251)

[2.2.13 Структура данных алгоритма PageControlChanging 31](#_Toc135979252)

[2.2.14 Структура данных алгоритма LVPersonsSelectItem 31](#_Toc135979253)

[2.2.15 Структура данных алгоритма ReadPersonsFromFile 32](#_Toc135979254)

[2.2.16 Структура данных алгоритма AmountOfGen 32](#_Toc135979255)

[2.2.17 Структура данных алгоритма WriteIntoFile 33](#_Toc135979256)

[2.2.18 Структура данных алгоритма СreateLinkedListFromPersons 33](#_Toc135979257)

[2.3 Схема алгоритмов решения задачи по ГОСТ 19.701-90 34](#_Toc135979258)

[2.3.1 Схема алгоритма GoingThroughTree 34](#_Toc135979259)

[2.3.2 Схема алгоритма GetFullName 36](#_Toc135979260)

[2.3.3 Схема алгоритма ActLoadPictureExecute 37](#_Toc135979261)

[2.3.4 Схема алгоритма RenamePhotoPath 38](#_Toc135979262)

[2.3.5 Схема алгоритма LVPersonsSelectItem 39](#_Toc135979263)

[2.3.6 Схема алгоритма SBPersonsFind 40](#_Toc135979264)

[2.3.7 Схема алгоритма FormClose 42](#_Toc135979265)

[2.3.8 Схема алгоритма ReadPersonsFromFile 43](#_Toc135979266)

[2.3.9 Схема алгоритма WriteIntoFile 44](#_Toc135979267)

[2.3.10 Схема алгоритма Randoming 45](#_Toc135979268)

[2.4 Графический интерфейс 46](#_Toc135979269)

[2.4.1 Описание графических компонентов формы Work 46](#_Toc135979270)

[2.4.2 Описание графических компонентов формы DeletingPerson 49](#_Toc135979271)

[2.4.3 Описание графических компонентов формы Help 50](#_Toc135979272)

[3 Тестирование и проверка работоспособности программного средства 51](#_Toc135979273)

[3.1 Запуск программы 51](#_Toc135979274)

[3.1.1 Тест 1 51](#_Toc135979275)

[3.2 Работа с персонами 51](#_Toc135979276)

[3.2.1 Тест 2 51](#_Toc135979277)

[3.2.2 Тест 3 52](#_Toc135979278)

[3.2.3 Тест 4 53](#_Toc135979279)

[3.2.4 Тест 5 53](#_Toc135979280)

[3.3 Отрисовка дерева 54](#_Toc135979281)

[3.3.1 Тест 6 54](#_Toc135979282)

[3.3.2 Тест 7 55](#_Toc135979283)

[3.4 Экспорт дерева 55](#_Toc135979284)

[3.4.1 Тест 8 55](#_Toc135979285)

[3.5 Создание и открытие типизированных файлов 56](#_Toc135979286)

[3.5.1 Тест 9 56](#_Toc135979287)

[3.5.2 Тест 10 56](#_Toc135979288)

[4 Руководство по установке 57](#_Toc135979289)

[4.1 Минимальные системные требования 57](#_Toc135979290)

[4.2 Установка 57](#_Toc135979291)

[4.3 Работа с приложением 60](#_Toc135979292)

[Заключение 63](#_Toc135979293)

[Список литературы 64](#_Toc135979294)

[Приложение А 65](#_Toc135979295)

[Приложение Б 87](#_Toc135979296)

[Приложение В 94](#_Toc135979297)

[Приложение Г 100](#_Toc135979298)

Введение

Данный курсовой проект посвящен программному средству "Создание генеалогических деревьев".

Генеалогические деревья широко используются для исследования семейной истории и восстановления родословной. Они позволяют установить связи между различными членами семьи и визуализировать их родственные отношения.

Основная функция программного средства "Создание генеалогических деревьев" заключается в создании и визуализации генеалогических деревьев. Пользователь может добавлять информацию о родственниках, указывать их связи и степень родства. Программа позволяет отображать генеалогическое древо в удобном и понятном виде.

В дополнение к основной функции, программное средство может предоставлять и другие возможности, такие как сохранение и загрузка созданных деревьев, поиск данных, добавление фотографий и дополнительной информации о родственниках.

Целью данной работы является разработка программного средства "Создание генеалогических деревьев", а также создание документации в виде пояснительной записки, которая будет содержать описание функций программы, алгоритмов работы, структуры данных, тестовых сценариев и другую необходимую информацию.

В результате выполнения курсового проекта ожидается получение полнофункционального и удобного в использовании программного средства, способного упростить работу с генеалогическими данными и помочь пользователям в исследовании своей семейной истории.

# Анализ литературных источников

## Анализ существующих аналогов

Долгое время для создания генеалогических, или семейных, древ использовался примитивный метод – рисование на бумажном носителе. Однако такой экземпляр будет единственным в своём роде, что не позволяет обмениваться им с кем-то из своих родственников. Помимо этого такой способ не нёс никакой дополнительной информации, кроме как ФИО, дат рождения и смерти и родственных связей, и был уязвим перед потерей и уничтожением. Но с приходом технологий большинство перечисленных проблем исчезли: появилась возможность создавать бесчисленное количество копий без особых затрат сил, добавлять различные фотографии, добавлять большое количество персон и их характеристик и т.д.

В данный момент существует не так много действительно хороших сервисов, которые могли бы предоставить широкий набор возможностей для создания столь важной для каждой большой семьи вещи. На рассмотрения были выбраны следующие сервисы: «MyHeritage» и «Древо жизни». При обзоре существующих аналогов были оценены их возможности, достоинства и недостатки.

### Программное средство «MyHeritage»

«MyHeritage» — семейно-ориентированная социальная сеть и генеалогический сайт, который позволяет зарегистрированным участникам создавать собственные семейные веб-сайты, обмениваться фотографиями и видео, организовывать семейные праздники, создавать родовые древа и искать предков, производить генеалогические ДНК-тесты. «Моё наследие» (MyHeritage) насчитывает более 71 000 000 пользователей и является одним из крупнейших сайтов в области социальных сетей и генеалогии. В связи с этим существует как веб-сайт (myheritage.com), так и программа (Family Tree Builder) для создания генеалогических древ от этого сервиса.

Существенным различием является дизайн и функционал: программа Family Tree Builder намного хуже в этих характеристиках.

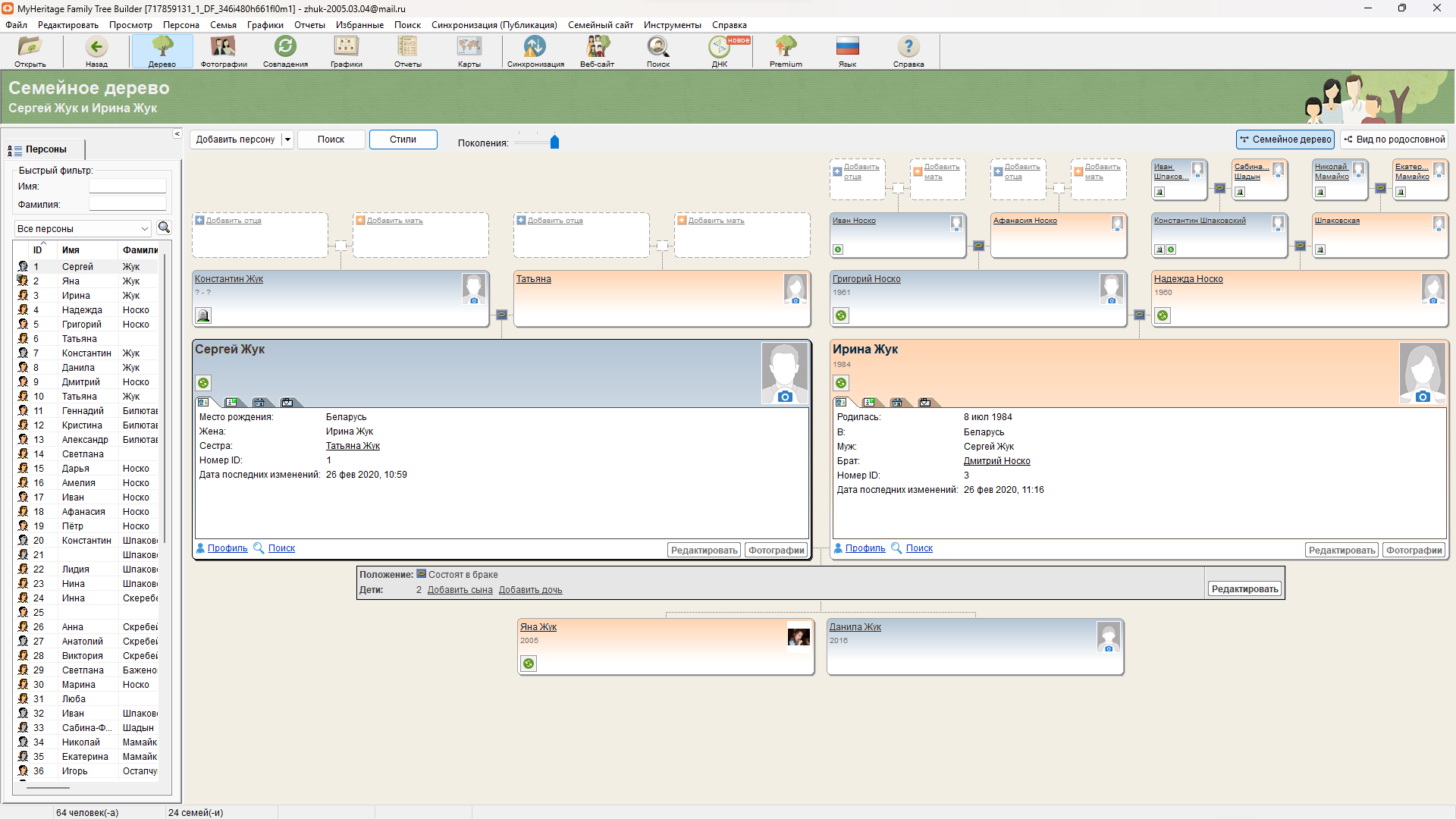


Рисунок 1 – Окно дерева в программе Family Tree Builder

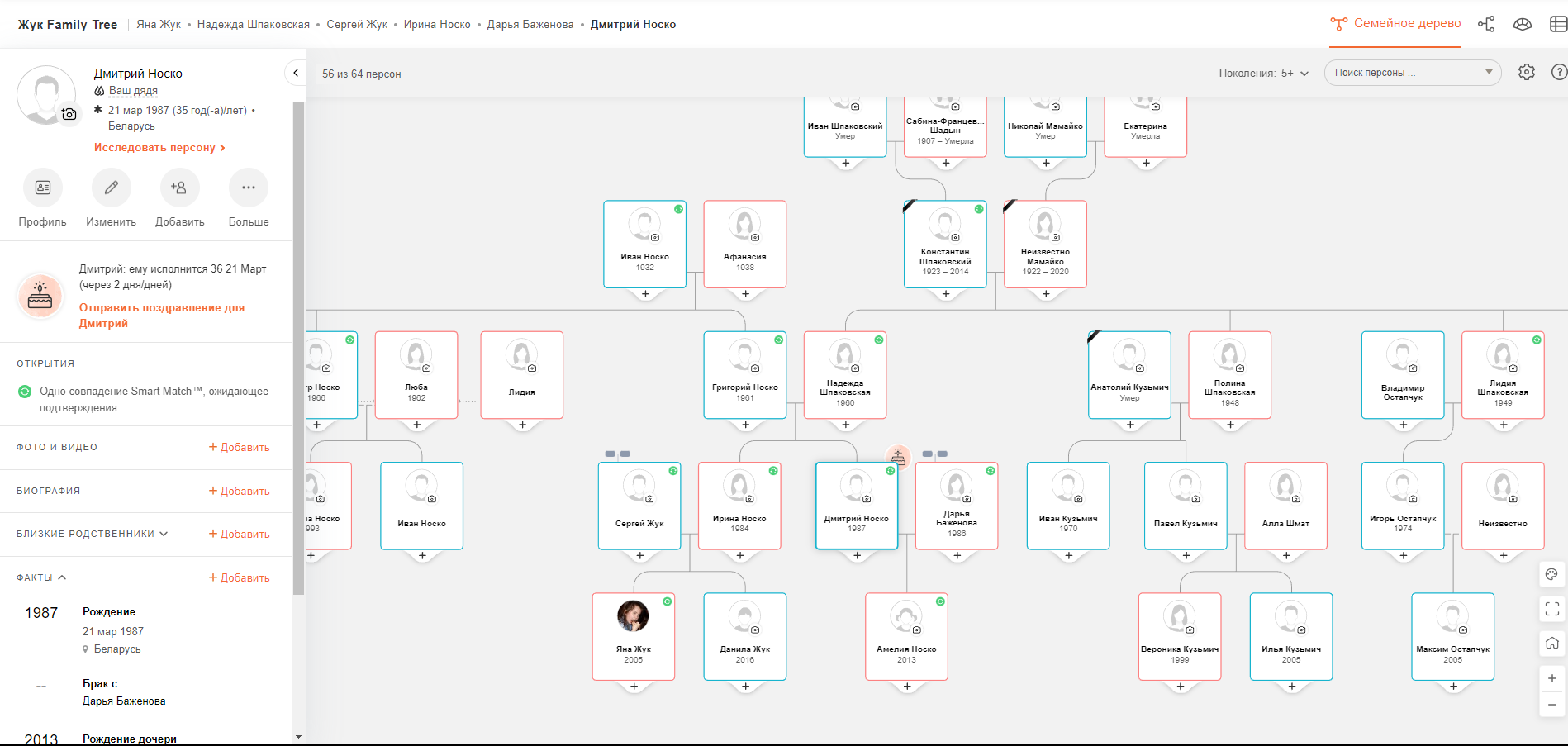


Рисунок 2 – Окно дерева на веб-сайте myheritage.com

Основными и общими функция для данных продуктов сервиса «MyHeritage» являются:

* составление и просмотр семейных древ различного вида и оформления;
* поиск конкретной персоны среди всех добавленных;
* печать семейных древ;
* использование различных источников, предоставляемых по подписке;
* добавление фотографий;
* создание и печать графиков и отчётов различных типов.

Рассмотрим подробнее самую важную функцию – создание древа, – которая подразделяется на множество других возможностей:

* добавление и удаление персон;
* добавление информации о них:
  + краткая биография;
  + место проживания, контактная информация (если жива);
  + родственные связи;
  + сфера деятельности, обучение, звания;
  + место рождения, смерти, захоронения;
  + даты рождения и смерти;
  + факты
  + пол;
* добавление фотографий;
* установка родственных связей (родитель – ребёнок, муж – жена) и их уточнение (бывший супруг, приёмный родитель и т.п.).

Вся информация о конкретной персоне отображается в его личной карточке:

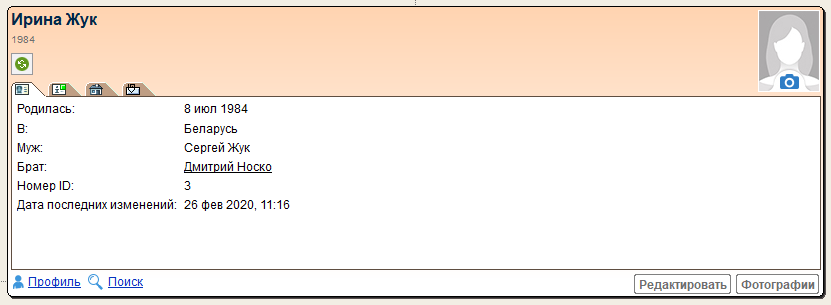


Рисунок 3 – Карточка персоны в программе Family Tree Builder

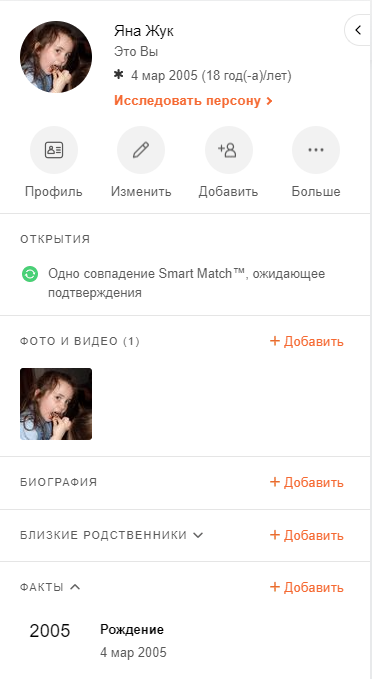


Рисунок 4 – Карточка персоны на веб-сайте myheritage.com

Существенными достоинствами продуктов продуктов сервиса «MyHeritage» можно назвать:

* интуитивно понятный интерфейс;
* автоматическое определение родственных связей (сестра, брат, тётя, бабушка и т.д.) на основе базисных (родитель – ребёнок, муж – жена);
* синхронизация с аккаунтом;
* слияние с другими деревьями;
* экспорт дерева в виде рисунка;
* уточнение информации при использовании предоставляемых источников;
* экспорт-импорт данных в общепринятый формат генеалогических программ GEDCOM.

Недостатками же можно назвать следующие пункты:

* малый функционал у бесплатной версии;
* ориентированность на западный рынок, в связи с чем возникают сложности с кириллицей и малое количество документов и источников из бывших стран СССР;
* слабая техническая поддержка, особенно у бесплатной версии;
* нет возможности просмотреть всё древо целиком, если количество персон по одной линии больше, чем по второй.

### Программное средство «Древо жизни»

Многие люди интересуются своей историей, историей своей семьи. Собрать минимум информации, поговорить со старшим поколением, изучить фотографии и документы из семейного архива может каждый. Программа «Древо Жизни» выполнит остальную работу: хранение, систематизацию и отображение собранной информации, в том числе в виде генеалогического дерева (родословного древа).

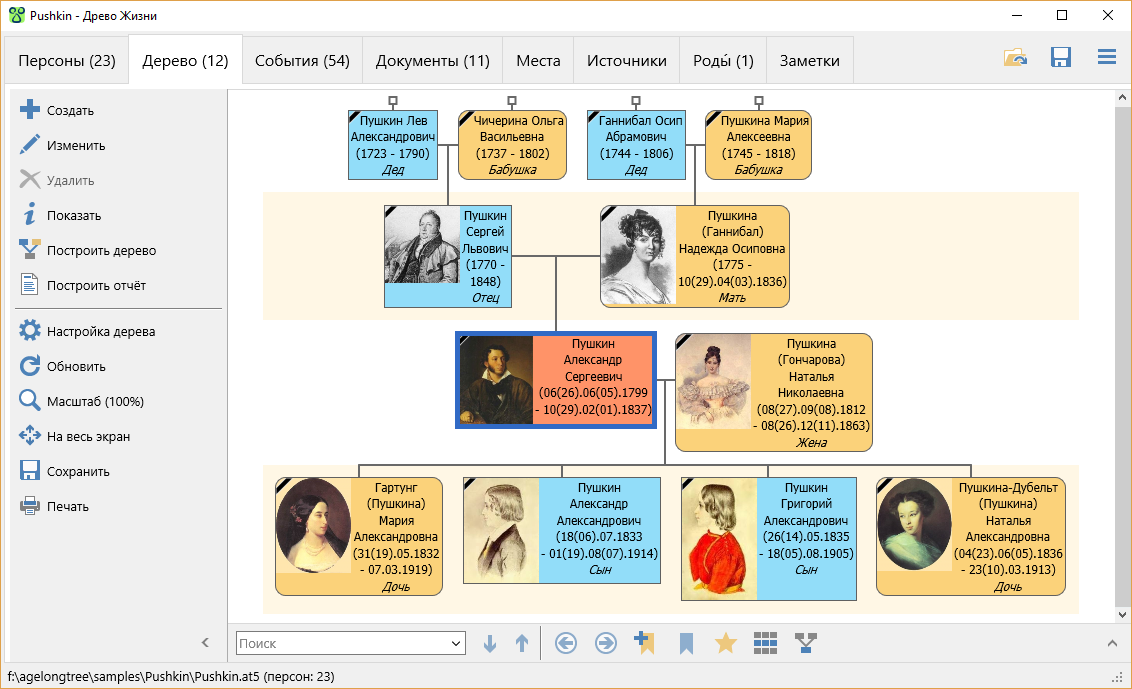


Рисунок 5 – Окно древа в программе «Древо жизни»

Программное средство «Древо Жизни» схоже по функционалу с «MyHeritage» и имеет следующие возможности:

* построение и просмотр генеалогического древа любого типа (только прямые родственники, прямые родственники с братьями и сёстрами, кровные родственники, все родственники) для любой выбранной персоны;
* добавление фотографий;
* составление статистик различных видов;
* печать семейных древ;
* сохранение древа в различных форматах для последующего просмотра;
* автоматически напоминает о необходимости создать резервную копию данных на внешнем диске — для предотвращения потери данных при поломке компьютера\флешки;
* хранит все вводимые данные только локально, то есть на компьютере пользователя, не загружая их в интернет.

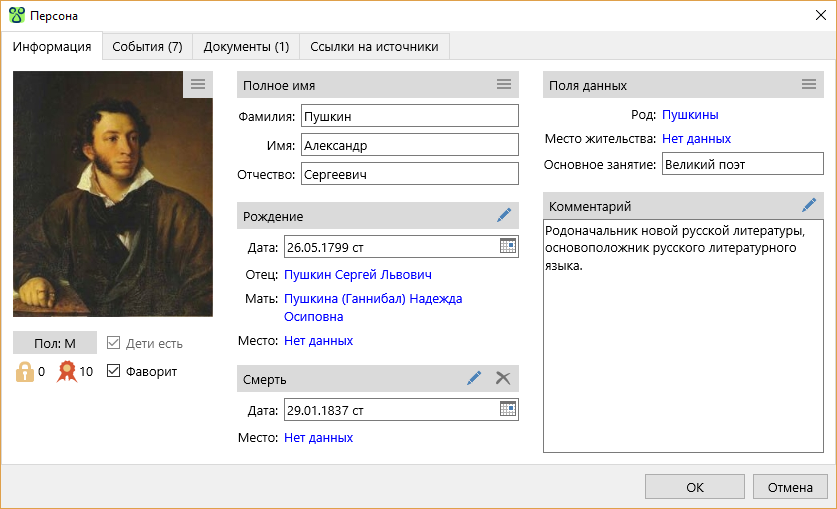


Рисунок 6 – Карточка персоны в программе «Древо жизни»

В связи с тем, что данное программное средство крайне схоже с продуктами «MyHeritage», то оно имеет также схожие преимущества:

* интуитивно понятный интерфейс;
* автоматическое определение родственных связей (сестра, брат, тётя, бабушка и т.д.) на основе базисных (родитель – ребёнок, муж – жена);
* возможность просмотра древа целиком;
* ориентированность на пользователя из СНГ, что препятствует возникновению проблем с кириллицей;
* экспорт дерева в виде рисунка в различных расширениях;
* экспорт-импорт данных в общепринятый формат генеалогических программ GEDCOM.

Так как данная программа предназначена для использования на персональном компьютере, то можно выделить следующие недостатки:

* невозможность просмотра древ на мобильных устройствах;
* просмотр древа возможен только на носителе
* ограничение в количестве персон;
* отсутствие слияния с другими древами.

### Итоги анализа существующих аналагов

Таким образом, практически все программные средства для создания, просмотра и печати генеалогических древ имеют схожий функционал, дизайн, преимущества и недостатки. Немаловажной частью программ данной области является экспорт и импорт в общепринятый формат GEDCOM, создание и экспорт родословного дерева в различные форматы для просмотра и печати, создание родственных связей и различных характеристик персон.

## Анализ средств разработки программного средства

Полагается, что данное программное средство помимо своего основного назначения – создание семейных древ – будет выполнять некоторые дополнительные функции, такие как, например, создание экспорт рисунка семейного древа, поиск персон в данном дереве, их сортировка и фильтрация по различным параметрам, редактирование уже созданного древа.

В данном программном средстве будут использоваться следующие основные структуры данных:

* записи;
* деревья;
* линейные однонаправленные списки;
* файлы.

### Запись

Запись (record) - это структура данных, которая позволяет объединять несколько значений разных типов в одну переменную. Каждое значение в записи представляется отдельным полем (field), которое может иметь свой тип данных. Например, поле может быть целочисленным, вещественным, логическим, символьным, строковым, указателем на объект, и т.д.

Запись может использоваться для создания сложных структур данных, таких как массивы или связанные списки, а также для представления состояния объектов и данных в приложениях. В записи можно хранить данные о пользователе, данные о продукте, данные о заказе, и многое другое.

Эта структура данных имеет ряд преимуществ, таких как:

1. Удобство доступа к полям: каждое значение в записи представляется в виде отдельного поля, что позволяет легко получать и изменять значения внутри записи.
2. Легкость передачи в функции и процедуры: записи могут передаваться в качестве параметров в функции и процедуры, что позволяет передавать сложные структуры данных и работать с ними как с единой переменной.
3. Возможность использования записей для создания сложных структур данных: записи можно использовать для создания массивов или связанных списков, что позволяет создавать сложные структуры данных в программе.

### Деревья

Дерево - это структура данных, состоящая из узлов, связанных между собой в виде иерархической структуры. Каждый узел может иметь несколько дочерних узлов, но только один родительский узел, кроме корневого узла, который не имеет родительского узла. Каждый узел может содержать некоторые данные, которые могут быть использованы для представления информации или значений. Деревья широко используются для представления иерархических отношений, например, для представления файловой системы, иерархии сайта, структуры данных и т.д.

### Линейные однонаправленные списки

Линейный односвязный список - это структура данных, которая состоит из элементов, каждый из которых содержит какое-то значение и ссылку на следующий элемент в списке. Первый элемент списка называется головой (head), а последний элемент не содержит ссылки на следующий элемент и называется хвостом (tail). Элементы списка хранятся в памяти не последовательно, а связаны между собой ссылками, что позволяет быстро добавлять и удалять элементы из списка. Для доступа к элементам списка необходимо начинать с головы и последовательно переходить по ссылкам на следующий элемент, пока не будет достигнут нужный элемент.

Линейный односвязный список может использоваться для хранения данных, которые могут быть упорядочены, но при этом могут изменяться в процессе выполнения программы. Например, список может использоваться для хранения списка задач, имен и контактов людей, и т.д.

Преимущества линейного односвязного списка включают возможность быстрой вставки и удаления элементов из середины списка, а также возможность изменения размера списка во время выполнения программы.

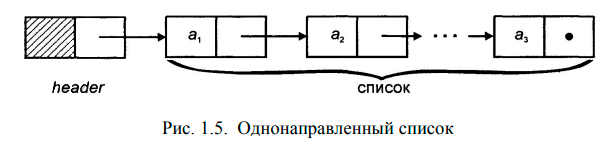


Рисунок 1 – Графическое представление линейного односвязного списка

### Файлы

Понятие файла может быть представлено двумя терминами: физический и логический файлы. Физический файл – это поименованная область памяти на внешнем носителе, в которой хранится некоторая информация. Логический файл – представление физического файла в программе.

Логические и физические файлы тесно связаны друг с другом. Логический файл описывает формат и структуру данных, которые он содержит, а физический файл представляет собой место хранения этих данных на устройстве. Когда данные записываются в логический файл, они сохраняются в соответствующем физическом файле, который затем может быть прочитан для получения этих данных. При чтении данных из логического файла, система оперирует на физическом файле, считывая данные из определенного участка жесткого диска, и затем интерпретирует их в соответствии с форматом логического файла.

Существует три типа переменных файла:

* текстовые файлы;
* типизированные файлы;
* не типизированные файлы.

Текстовый файл представляет собой последовательность символов, которая может быть разделена на строки. Текстовые файлы могут быть открыты в одном из двух режимов: для чтения или для записи. Кроме того, текстовые файлы могут быть открыты в режиме добавления, который позволяет добавлять данные в конец файла без удаления его содержимого. Файл, открытый в режиме добавления, всегда открывается для записи, но указатель позиции устанавливается на конец файла.

Типизированные файлы связываются с файловыми переменными, объявленными как "file of <Тип>". Файл считается состоящим из элементов, каждый из которых имеет тип <Тип>.

Не типизированные файлы могут быть связаны только с файловыми переменными, которые были объявлены как "file". Файл считается состоящим из элементов, размер которых определяется при открытии файла.

Существуют следующие способы доступа файла:

* последовательный доступ;
* прямой доступ.

При последовательном доступе по файлу можно двигаться только последовательно, начиная с первого его элемента. В этом случае доступен лишь очередной элемент файла. Чтобы добраться до n-го элемента файла, необходимо начать с первого элемента и пройти через предыдущие n – 1 элементов.

При прямом доступе можно обратиться непосредственно к элементу файла с номером n, минуя предварительный просмотр n – 1 элемента файла.

Важно отметить, что для работы с файлами в программировании нужно уметь открывать и закрывать файлы, читать и записывать данные в файлы, а также обрабатывать ошибки, связанные с файлами, такие как отсутствие файла или ошибка доступа. Также нужно следить за использованием ресурсов компьютера при работе с файлами, чтобы избежать проблем с памятью или производительностью.

Можно сделать вывод о том, что файлы являются важными элементами программирования, которые позволяют хранить и загружать данные в различных форматах. Файлы позволяют хранить данные между сеансами работы программы, предоставляют способ сохранения и загрузки больших объемов данных, могут использоваться для обмена данными между различными приложениями и системами и обеспечивают сохранность информации в случае сбоев в работе программы или выключения компьютера.

### Сортировка с помощью разделения

QuickSort (быстрая сортировка) - это алгоритм сортировки, который использует принцип "разделяй и властвуй", является существенно улучшенным вариантом алгоритма сортировки с помощью прямого обмена известного в том числе своей низкой эффективностью. Он работает путем выбора элемента из списка в качестве опорного (pivot), а затем разбивает список на две части: элементы, меньшие опорного, и элементы, большие опорного. Затем алгоритм рекурсивно применяется к каждой из двух частей списка до тех пор, пока не будет достигнут базовый случай, когда список содержит один элемент.

В Quicksort исходят из того соображения, что для достижения наилучшей эффективности сначала лучше производить перестановки на большие расстояния. Предположим, у нас есть n элементов, расположенных по ключам в обратном порядке. Их можно отсортировать за n/2 обменов, сначала поменять местами самый левый с самым правым, а затем последовательно двигаться с двух сторон. Это возможно только в том случае, когда мы знаем, что порядок действительно обратный.

Суть итеративного решения заключается во введении списка требуемых разделений, т.е. разделений, которые необходимо провести. На каждом этапе возникают две задачи по разделению. И только к одной из них можно непосредственно сразу же приступить в очередной итерации, другая же заносится в упомянутый список. При этом существенно, что требования из списка выполняются несколько специфическим образом, а именно в обратном порядке. Следовательно, первое из перечисленных требований выполняется последним, и наоборот. В нерекурсивной версии Quicksort каждое требование задается просто левым и правым индексами – это границы части, требующей дальнейшего разделения. Таким образом, вводится переменная-массив под именем Stack и индекс S, указывающий на самую последнюю строку в стеке. Далее выполняется основной цикл, который извлекает пару индексов из стека и сортирует соответствующий подсписок. Если размер подсписка больше одного элемента, он разбивается на две части с помощью выбранного опорного элемента и индексы этих двух подсписков добавляются в стек.

Алгоритм продолжает работу до тех пор, пока стек не будет пустым, что означает, что все подсписки были отсортированы.

## Спецификация функциональных требований

Требование – описание того, какие функции и с соблюдением каких условий должна выполнять приложение в процессе решения полезной для пользователя задачи. Целью программного обеспечения является на основе сравнения достоинств и недостатков представленных ранее существующих аналогов реализовать с получением наилучшего результата.

Функциональное требование – это заявление о том, как должна вести себя система. Он определяет, что система должна делать, чтобы удовлетворить потребности или ожидания пользователя. Функциональные требования можно рассматривать как функции, которые обнаруживает пользователь. В такой предметной области, как создание генеалогических древ, основные функциональные требования связаны с управлением и редактированием информации о персонах в деревьях пользователя.

Таблица 1 – Функциональные требования

|  |  |
| --- | --- |
| Идентификатор | Требование |
| ФТ-1 | Создание генеалогического древа |
| ФТ-2 | Просмотр генеалогического древа |
| ФТ-3 | Создание персон |
| ФТ-4 | Удаление персон из древа |
| ФТ-5 | Установка родственных связей |
| ФТ-6 | Редактирование информации о персоне |
| ФТ-7 | Вывод списка всех персон в генеалогическом древе |
| ФТ-8 | Представление генеалогического древа в графическом виде |
| ФТ-9 | Вывод списка всех персон в генеалогическом древе в отсортированном виде |
| ФТ-10 | Поиск персон по определенным параметрам |
| ФТ-11 | Экспорт графического представления генеалогического древа в формате JPEG, PNG |

Комментарии к ФТ-1.

ФТ-1 предполагает возможность создания файла для нового генеалогического древа.

Комментарии к ФТ-2.

ФТ-2 предполагает возможность открытия файла с уже существующим генеалогическим древом.

Комментарии к ФТ-7.

ФТ-7 предполагает вывод в виде списка с колонками информации всех персон текущего генеалогического древа.

Комментарии к ФТ-8.

ФТ-8 предполагает возможность просмотра текущего генеалогического древа в графическом представлении, корень которого пользователь выбирает посредством выбора персоны в общем списке.

Комментарии к ФТ-10.

ФТ-10 позволяет произвести поиск по какому-либо параметру, а конкретно по фамилии, по имени, по дате рождения или дате смерти, персоны и вывести пользователю найденный список персон.

Комментарии к ФТ-11.

ФТ-11 позволяет экспортировать графическое представление в такие расширения, как JPEG и PNG.

# Разработка и проектирование программного средства

## Описание алгоритмов решения задачи

Таблица 2.1 – Описание алгоритмов решения задачи

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| №  п.п. | Наименование алгоритма | Название алгоритма | Формальные  параметры | Предполагаемый тип реализации |
| 1 | Инициализация главной формы | Представление пользователю возможности выбора манипуляции над генеалогическим древом; вывод результата.  Вызов следующей подпрограммы:  PageControl  Change,  FormActivate,  PageControl  Changing |  |  |
| 2 | PageControlChange  (  Sender  ) | Контроль изменения пользователем вкладки и в соответствии с тем, какая вкладка открылась, выполнение определенных действий | Sender – получает от фактического параметра адрес | Процедура |
| 3 | ActDrawTreeExecute  (  Sender  ) | Отрисовка генеалогического древа от корня в соответствии со значением масштаба и количеством поколений | Sender– получает от фактического параметра адрес | Процедура |

Продолжение Таблицы 2.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 4 | ActOpenPersonCard  Execute  (  Sender  ) | Открытие карточки персоны и вывод информации на форму о ней в соответствии с выбранной персоной | Sender – получает от фактического параметра адрес | Процедура |
| 5 | FormActivate  (  Sender  ) | Выполнение определенных действий, таких как вывод начальной информации в карточке персоны | Sender – получает от фактического параметра адрес | Процедура |
| 6 | AddToTheList  (  Person,  NeededList  ) | Добавление персоны в список NeededList | Person, NeededList – получает от фактического параметра адрес | Процедура |
| 7 | ShowInfoInCard  (  Person  ) | Показ информации о персоне в карточке персоны, если она выбрана | Person – получает от фактического параметра адрес | Процедура |
| 8 | AddToTheCombos  (  Person  ) | Добавление существующих персон в выпадающие списки | Person – получает от фактического параметра адрес | Процедура |
| 9 | AmountOfBlocks  (  WWidth,  WHeight,  CWidth,  CHeight,  Margins,  Res  ) | Подсчет максимального количества блоков персон с размерами CWidth, CHeight и границами Margins, которые можно отрисовать в окне с размерами  WWidth, WHeight | WWidth – получает от фактического параметра адрес;  WHeight, CWidth, CHeight,  Margins – получает от фактического параметра значение; | Функция. Res – возвращаемый параметр |

Продолжение Таблицы 2.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  | Res – получает от фактического параметра адрес |  |
| 10 | PageControlChanging  (  Sender,  AllowChange  ) | Контроль изменения пользователем вкладки и в соответствии с тем, какая вкладка закрылась, выполнение определенных действий.  Вызов следующих подпрограмм:  RenamePhotoPath,  AddToTheCombos,  AddToTheList,  ChangeValueInCombos,  ResetControls | Sender – получает от фактического параметра адрес;  AllowChange –  получает от фактического параметра адрес | Процедура |
| 11 | LVPersonsSelectItem  (  Sender,  Item,  Selected  ) | Присваивание главной персоне значение выбранной персоны в списке | Sender, Item – получает от фактического параметра адрес;  Selected – получает от фактического параметра значение | Процедура |
| 12 | GetFullName  (  Person,  Name  ) | Получение ФИО выбранной персоны | Person, Name – получает от фактического параметра адрес | Функция.  Name – возвращаемое значение |

Продолжение Таблицы 2.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 13 | GoingThrough  Tree  (  HeadOfTree,  Visited,  Processes  ) | Проходка через всё n-арное дерево, представленное однонаправленным списком с «головой» HeadOfTree, и совершение над каждым элементом метода из массива методов Processes | HeadOfTree – получает от фактического параметра адрес;  Visited – получает от фактического параметра адрес;  Process – получает от фактического параметра адрес | Процедура |
| 14 | RenamePhotoPath  (  NewString,  Path  ) | Переименовывает путь фотографии Path на NewString | Path – получает от фактического параметра адрес;  NewString – получает от фактического параметра копию значения | Процедура |
| 15 | TFwork.SBPersons  Find  (  Sender  ) | Производит поиск в списке персон и выводит результат поиска.  Вызов следующей подпрограммы:  AddToTheList | Sender – получает от фактического параметра адрес | Процедура |

Продолжение Таблицы 2.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 16 | RemovePerson  (  Person,  WayOfRemoving  ) | Удаление персоны Person путём, который выберет пользователь на всплывающей форме | Person – получает от фактического параметра копию значения;  WayOf  Removing – получает от фактического параметра копию значения | Процедура |
| 17 | TFwork.TBtnDeleteClick  (  Sender  ) | Открывает всплывающую форму для получения способа удаления персоны как родителя-одиночку и уточняет выбор удаления | Sender – получает от фактического параметра адрес | Процедура |
| 18 | ReadPersonsFrom  File  (  FileName,  Person  ) | Прочитывает из файла FileName список персон.  Вызов следующей подпрограммы:  CreateLinkedListFromPersons | FileName, Person ­– получает от фактического параметра адрес | Функция. Person – возвращаемый параметр |
| 19 | AmountOfGen  (  Gen  Root,  Persons,  Index,  Res  ) | Рекурсивно рассчитывает количество персон Res в поколении Gen и записывает их указатели в Persons. Начало отсчета – персона с указателем Root | Res, Persons ­– получает от фактического параметра адрес;  Gen, Root, Index – получает от фактического параметра копию значения | Функция.  Res – возвращаемый параметр |

Продолжение Таблицы 2.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 20 | WriteIntoFile  (  StartPerson, NeededFileP  ) | Записывает в файл NeededFileP весь список персон, где корнем является StartPerson | StartPerson, NeededFileP – получает от фактического параметра копию значения | Процедура |
| 21 | CreateLinkedList  FromPersons  (  Persons,  Root  ) | Восстанавливает связи между прочитанными персонами Persons, где корень Root | Persons – получает от фактического параметра копию значения;  Root –получает от фактического параметра адрес | Процедура |

## Структура данных

### Структура типов основной программы

Таблица 2.2 – Структура типов основной программы

|  |  |  |
| --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение |
| PPerson | PPerson = ^TPerson | Указатель на персон |
| TLocation | TLocation = record  Country, City: String[15];  Address: String[30];  end | Хранение какого-либо адреса |
| TPerson | TPerson = record  Index: String[9];  Gender: TGender;  Mother, Father, Spouse,  StepM, StepF: PPerson;  Children: PPerson;  Adopted: Boolean;  Sibling: PPerson; | Хранение всей информации о персоне. Нефинализированная запись |

Продолжение Таблицы 2.2

|  |  |  |
| --- | --- | --- |
|  | POfBirth: TLocation;  FName, LName, Note:  String[100];  DOfBirth: TDateTime;  Photo: String;  case Alive: Boolean of  True:  (ContactInfo:  String[100]);  False:  (POfDeath: TLocation;  DOfDeath: TDateTime);  end |  |
| ProcessNode | procedure (HeadOf: PPerson) | Процедурный тип для обработки персон |
| Processes | array of ProcessNode | Массив всех процедур, необходимых для обработки |
| TFinPerson | TFinPerson = record  Index: String[9];  Gender: TGender;  Mother, Father, Spouse,  StepM, StepF: String[9];  Children: String[9];  Adopted: Boolean;  Sibling: String[9];  POfBirth: TLocation;  FName, LName, Note:  String[100];  DOfBirth: TDateTime;  Photo: String[255];  Alive: Boolean;  ContactInfo: String[100];  POfDeath: TLocation;  DOfDeath: TDateTime;  end | Перезапись всей информации о персоне для корректной записи в файл. Финализированная запись |
| NeededF | File of TFinPerson | Типизированный файл персон |

### Структура данных основной программы

Таблица 2.3 – Структура данных основной программы

|  |  |  |
| --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение |
| scale | Real | Текущий масштаб для отрисовки древа |
| CurrentPerson | PPerson | Текущая персона |
| NewPerson | Boolean | Была ли создана новая персона |
| CurrentChild | PPerson | Текущий ребенок |
| CurrPIndex | integer | Индекс текущей персоны в списке |
| PathForFileS, PathForFileR, nopicture, PathForPictures, PathForFile | string | Пути к файлу для сохранения, чтения, пустой картинки, к папке для картинок и файлов соответственно |

### Структура данных алгоритма AmountOfBlocks

Таблица 2.4 – Структура данных алгоритма AmountOfBlocks

(WWidth,WHeightCWidth, CHeight, Margins, Res)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| WWidth | Word | Ширина окна | Формальный |
| WHeight | Word | Высота окна | Формальный |
| Cwidth | Word | Ширина блока персоны | Формальный |
| CHight | Word | Высота блока персоны | Формальный |
| Margins | Word | Границы блока | Формальный |
| Res | byte | Максимальное количество блоков персоны, которое может вместится в окно | Формальный |

### Структура данных алгоритма ShowInfoInCard

Таблица 2.5 – Структура данных алгоритма ShowInfoInCard(Person)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Person | PPerson | Персона, информацию о которой необходимо показать в карточке | Формальный |

### Структура данных алгоритма AddToTheList

Таблица 2.6– Структура данных алгоритма AddToTheList(Person, NeededList)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Person | PPerson | Персона, которую необходимо добавить в список | Формальный |
| NeededList | TListView | Список, в который необходимо добавить | Формальный |
| PersonItem | TListItem | Элемент списка | Локальный |

### Структура данных алгоритма AddToTheCombos

Таблица 2.7 – Структура данных алгоритма AddToTheCombos(Person)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Person | PPerson | Персона, которую необходимо добавить во все выпадающие списки | Формальный |

### Структура данных алгоритма GetFullName

Таблица 2.8 – Структура данных алгоритма GetFullName(Person)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Person | PPerson | Персона | Формальный |
| Name | string | Её полное имя | Формальный |

### Структура данных алгоритма GoingThroughTree

Таблица 2.9 – Структура данных алгоритма GoingThroughTree(HeadOfTree, visited, Processes)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| HeadOfTree | PPerson | «Голова» списка | Формальный |
| Visited | TStringList | Список индексов персон, которые были посещены | Формальный |
| Processes | Process | Методы обработки каждой персоны | Формальный |
| ind | String | Индекс персоны | Локальный |

### Структура данных алгоритма RenamePhotoPath

Таблица 2.10 – Структура данных алгоритма RenamePhotoPath(NewString, Path)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| NewString | String | Новое имя файла | Формальный |
| Path | String | Текущее имя файла | Формальный |
| StartIndex, FinishIndex | Integer | Позиции, на которых нужно произвести замену | Локальный |
| Pict | File | Изменяемый файл | Локальный |

### Структура данных алгоритма TFwork.SBPersonsFind

Таблица 2.11– Структура данных алгоритма TFwork.SBPersonsFind(Sender)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Sender | TObject | Компонент, который вызвал | Формальный |
| searchQuery | String | Запрос | Локальный |
| searchLName, searchFName, searchDOfBirth, searchDOfDeath | Boolean | Параметры, по которым производится поиск | Локальный |
| Person | PPerson | Очередная персона для сравнения параметров | Локальный |
| i | Integer | Счетчик цикла | Локальный |

### Структура данных алгоритма RemovePerson

Таблица 2.12– Структура данных алгоритма RemovePerson(Person,

WayOfRemoving)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| WayOfRemoving | byte | Способ удаления связей с детьми, если родитель-одиночка | Формальный |
| Person | PPerson | Удаляемая персона | Формальный |
| NextNode, CurrNode | PPerson | Вспомогательные указатели | Локальный |

### Структура данных алгоритма ActDrawTreeExecute

Таблица 2.13– Структура данных алгоритма ActDrawTreeExecute (Sender)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Sender | TObject | Компонент, который вызвал | Формальный |
| WH, WW, dx, dy | Word | Высота и длина, Ох и Оу основного окна соответственно | Локальный |
| PersonCard, PCWork, Names, ImageR, Dates, WorkArea | TRect | Прямоугольники для корректной отрисовки | Локальный |
| NRectLeft, NRectBottom | Word | Вспомогательные переменные для координат | Локальный |
| i | Word | Высота блока персоны | Локальный |
| Margins | byte | Границы блока | Локальный |
| Generation, PersonInGen, CurrPersonInGen | byte | Поколение, максимальное количество блоков персоны, которое может вместится в окно, текущая персона соответственно | Локальный |
| ForOutput | string | Строка для отрисовки | Локальный |

Продолжение Таблицы 2.13

|  |  |  |  |
| --- | --- | --- | --- |
| ImageOfP | TPicture | Фото персоны для отрисовки | Локальный |
| ScaleGen | Real | Масштаб для текущего поколения | Локальный |
| NextPersons | ArrOfPersons | Массив персон, которые должны быть отрисованы | Локальный |

### Структура данных алгоритма PageControlChanging

Таблица 2.14 – Структура данных алгоритма PageControlChanging(Sender)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Sender | TObject | Компонент, который вызвал | Формальный |
| Position | byte | Вспомогательная переменная | Локальный |
| Change | Boolean | Стоит ли менять значения в выпадающих списках | Локальный |

### Структура данных алгоритма LVPersonsSelectItem

Таблица 2.15 – Структура данных алгоритма LVPersonsSelectItem(Sender, Item, Selected)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Sender | TObject | Компонент, который вызвал | Формальный |
| Item | TListItem | Выбранный элемент | Формальный |
| Selected | Boolean | Выбран ли элемент | Формальный |

### Структура данных алгоритма ReadPersonsFromFile

Таблица 2.16 – Структура данных алгоритма ReadPersonsFromFile(FileName, Res)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| FileName | String | Имя файла для чтения | Формальный |
| Res | PPerson | «Голова» полученного списка персон | Формальный |
| Persons | TPersonArray | Массив прочитанных персон | Локальный |
| PersonFile | NeededF | Файл для чтения | Локальный |
| PersonCount, i | Integer | Количество персон в файле и счетчик цикла | Локальный |

### Структура данных алгоритма AmountOfGen

Таблица 2.17 – Структура данных алгоритма AmountOfGen(Gen,Root, Persons, Index, Res)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Gen | Byte | Поколение для расчета | Формальный |
| Root | PPerson | Корень, для которого нужно рассчитать количество персон | Формальный |
| Persons | ArrOfPersons | Массив указателей на персоны, которые будут отрисованы | Формальный |
| Index | Byte | Индекс массива | Формальный |
| Res | Word | Количество людей для отрисовки | Формальный |
| Current  SubPerson | PPerson | Вспомогательный указатель | Локальный |

### Структура данных алгоритма WriteIntoFile

Таблица 2.18 – Структура данных алгоритма WriteIntoFile(StartPerson, NeededFileP)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| NeededFileP | String | Имя файла для записи | Формальный |
| StartPerson | PPerson | «Голова» списка персон для записи | Формальный |

### Структура данных алгоритма СreateLinkedListFromPersons

Таблица 2.19 – Структура данных алгоритма СreateLinkedListFromPersons

(Persons, Root)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Persons | TPersonArray | Массив персон | Формальный |
| Root | PPerson | «Голова» полученного списка персон | Формальный |
| i, g | Integer | Счетчики циклов | Локальный |
| Person, Personbuf | PPerson | Текущая персона для восстановления и персона для проверки на связь с текущей соответственно | Локальный |
| indexesOfAll | array of string | Массив всех индексов персон | Локальный |
| used | Boolean | Вспомогательная переменная | Локальный |

## Схема алгоритмов решения задачи по ГОСТ 19.701-90

### Схема алгоритма GoingThroughTree



Рисунок 2 – Схема алгоритма GoingThroughTree

(часть 1)



Рисунок 3 – Схема алгоритма GoingThroughTree

(часть 2)

### Схема алгоритма GetFullName



Рисунок 4 – Схема алгоритма GetFullName

### Схема алгоритма ActLoadPictureExecute



Рисунок 5 – Схема алгоритма ActLoadPictureExecute

### Схема алгоритма RenamePhotoPath



Рисунок 6 – Схема алгоритма RenamePhotoPath

### Схема алгоритма LVPersonsSelectItem



Рисунок 7 – Схема алгоритма LVPersonsSelectItem

### Схема алгоритма SBPersonsFind



Рисунок 8 – Схема алгоритма SBPersonsFind

(часть 1)



Рисунок 9 – Схема алгоритма SBPersonsFind

(часть 2)

### Схема алгоритма FormClose



Рисунок 10 – Схема алгоритма FormClose

### Схема алгоритма ReadPersonsFromFile



Рисунок 11 – Схема алгоритма ReadPersonsFromFile

### Схема алгоритма WriteIntoFile



Рисунок 12 – Схема алгоритма WriteIntoFile

### Схема алгоритма Randoming



Рисунок 13 – Схема алгоритма Randoming

## Графический интерфейс

Для организации графического интерфейса программного средства «Твои корни» была использована одна основная форма FWork, на которой располагается элемент PageControl, на котором располагаются вкладки, разделяющие программное средство на режимы взаимодействия с генеалогическим древо и персонами; и две дополнительные, всплывающие формы FDeletingPerson (удаление персоны) и FHelp (информация о программном средстве). Также использован компонент MainMenu для удобства взаимодействия с файлами.

### Описание графических компонентов формы Work

Форма Work – это основная форма, которая даёт пользователю доступ к основным функциям программного средства, таким как просмотр списка персон, добавление и удаление, поиск персон, просмотр информации о персонах и просмотр графического представление генеалогического древа, просмотр сведений о программе. Данная форма имеет вид, представленный на рисунках 14-16.

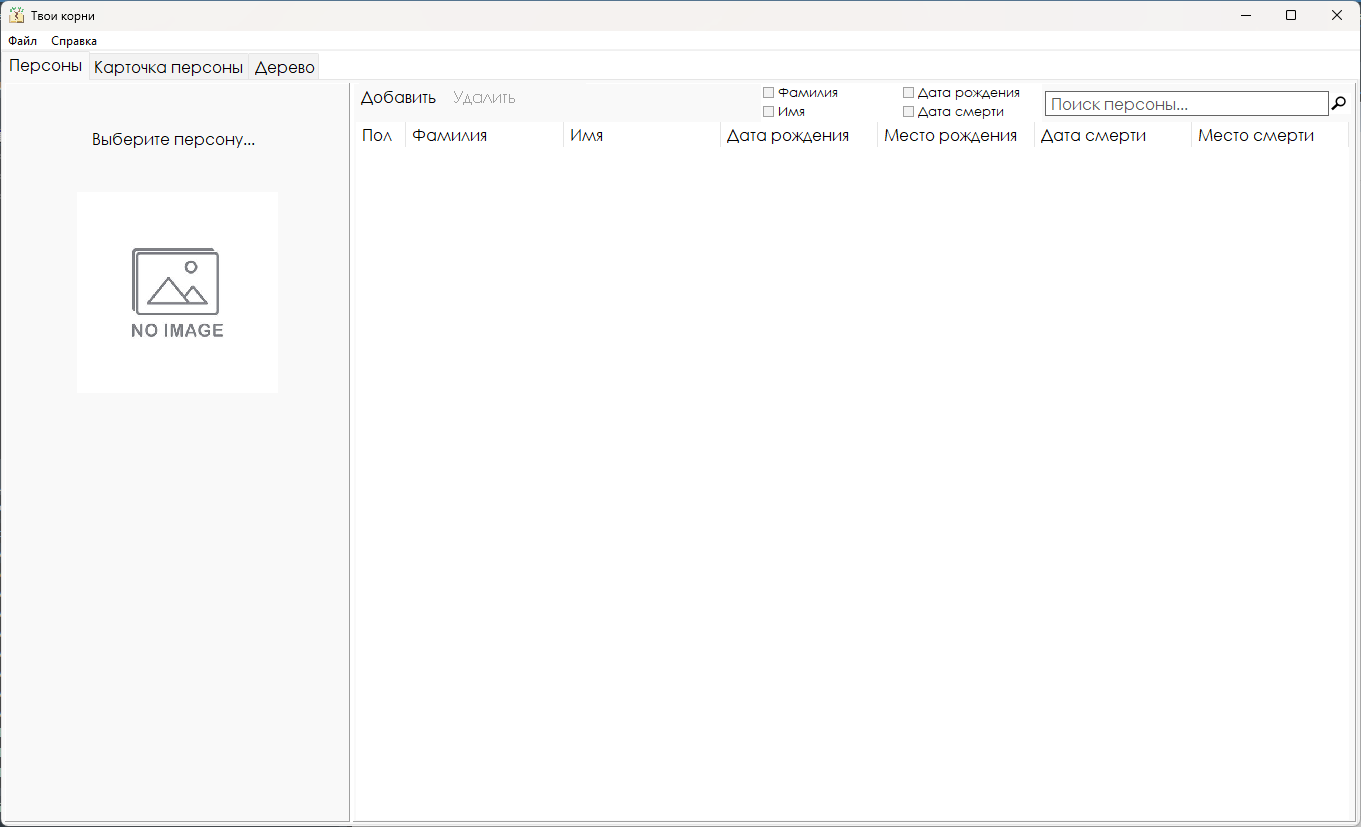


Рисунок 14 – Первая вкладка компонента PageControl

Вкладка 1: Персоны. Окно разделено на две части: левую, на которой отображено изображение выбранной персоны и её ФИО; на второй – список TListView всех персон и инструменты TToolButton взаимодействия с ними. Инструменты взаимодействия:

* добавить персону (открывает Вкладку 2 для добавления информации о новой персоне);
* удалить персону (недоступно, пока не выбрана персона в списке; удаляет выбранную персону с предварительным уточнением выбора, если персона – родитель-одиночка);
* поиск персоны по параметрам и параметры поиска (параметры, в которых производить поиск; например, слово Марина может быть фамилией и именем; необходимо кликнуть два раза на квадрат рядом с необходимым названием параметра, чтобы его использовать, третий клик отменяет выбор).

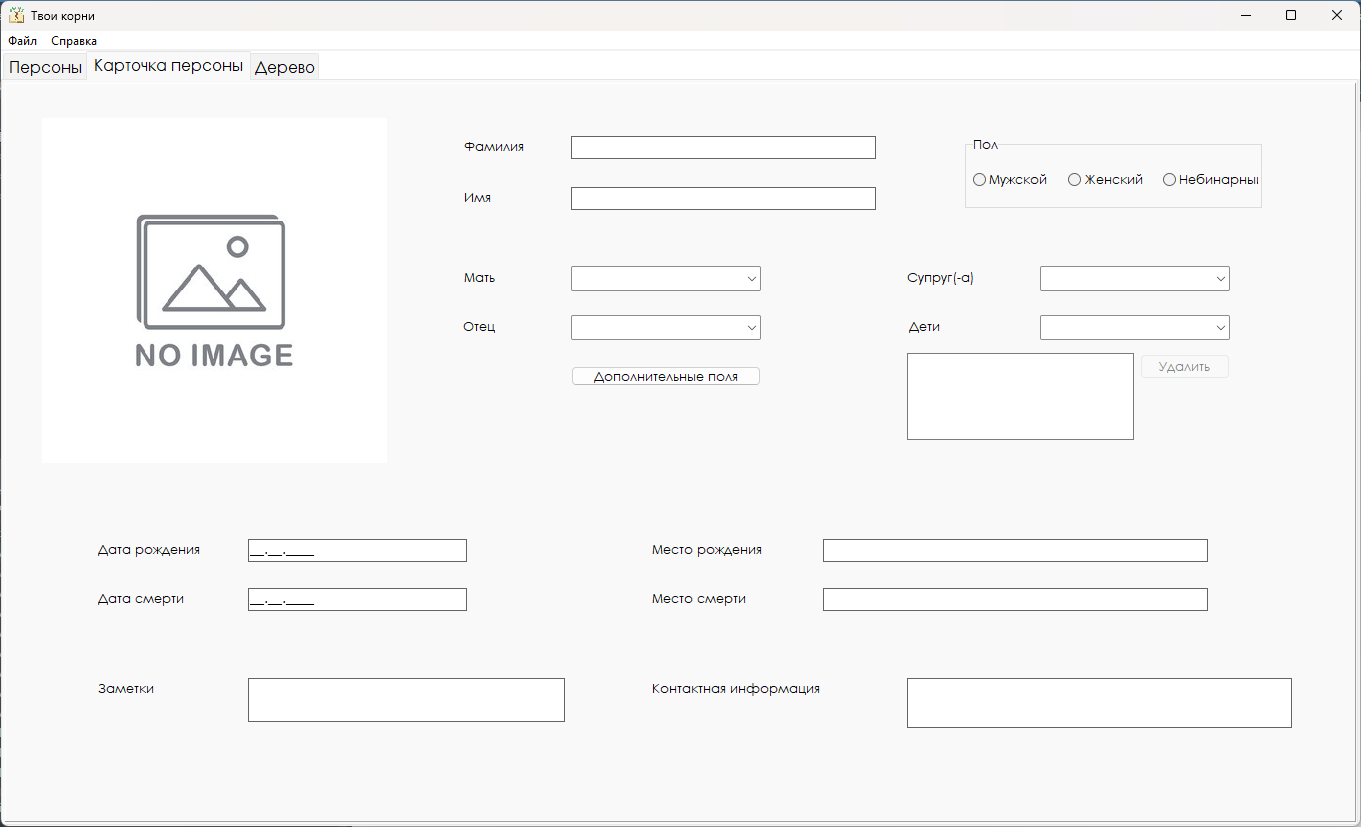


Рисунок 15 – Вторая вкладка компонента PageControl

Вкладка 2: Карточка персоны. В этом окне можно добавить/удалить/изменить информацию о выбранной персоне. Добавить связь с добавленными ранее персонами с помощью различных компонентов TComboBox, которые являются выпадающими списками и при вводе какого-либо параметра ФИО в поле добавления родственника автоматически будет открыт выпадающих список с подходящими персонами. Для удобства добавление детей происходит также, однако их отображение имеет другой вид: все дети персоны будут отображены в компоненте TListBox, который удобен в данной ситуации. Также рядом расположена кнопка удаления, которая станет активной после того, как будет выбран нужный ребенок. Для выбора пола персоны в правом верхнем углу находится компонент TRadioGroup, что позволяет выбрать только лишь одно значение пола. В верхнем левом угле отображено изображение персоны, при двойном клике на которую можно загрузить новое фото. При удалении родителя-одиночки решением может стать усыновление его детей вместо их удаления. В таком случае у усыновленных персон будут приемные родители, поменять/присвоить которых можно при нажатии на кнопку «Дополнительные поле».

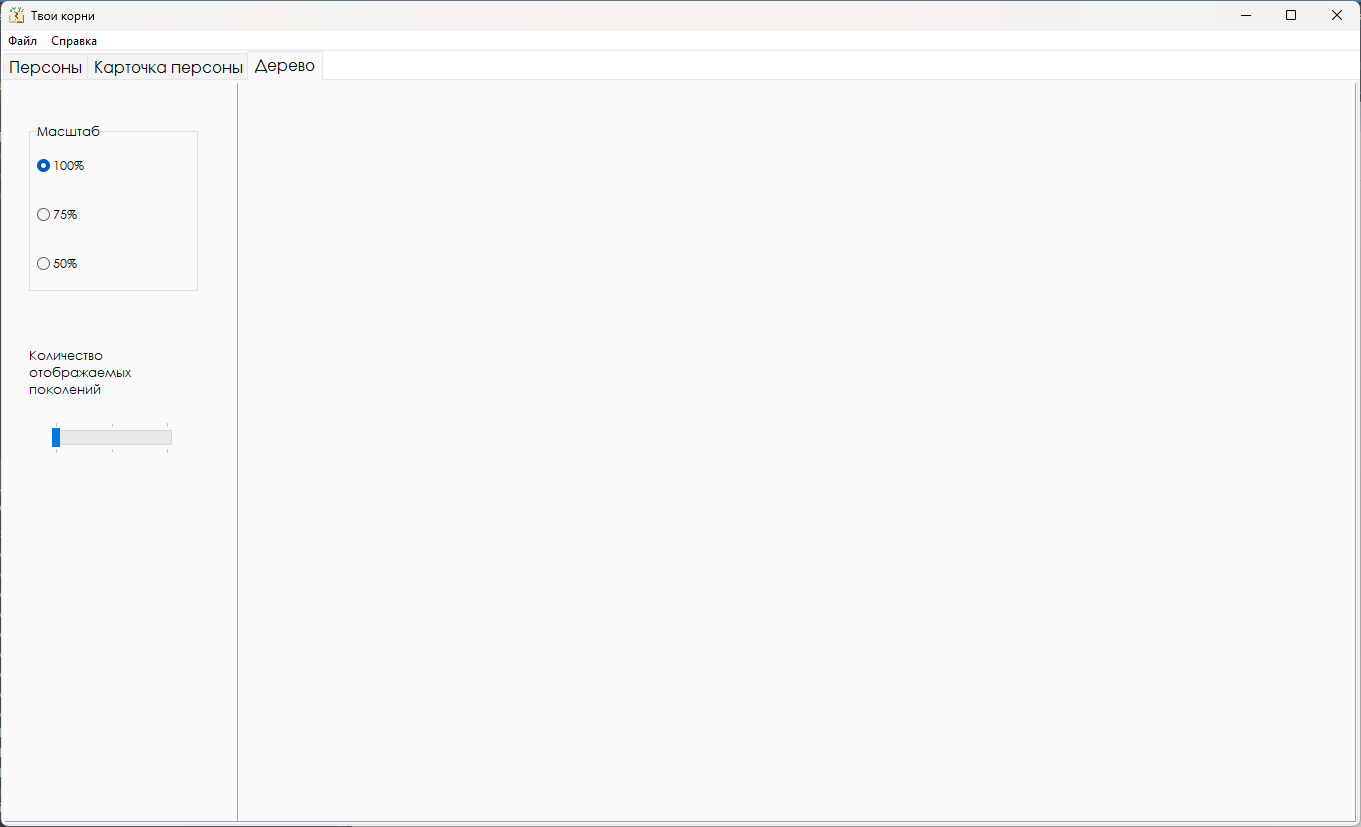


Рисунок 16 – Третья вкладка компонента PageControl

Вкладка 3: Дерево. Здесь можно получить графическое отображение созданного древа, корнем которого будет выбранная персона в списке Вкладка 1. Есть возможность настроить масштаб и количество отображаемых поколений вниз относительно корня древа. Кроме того, при отображении вместо подписи, указывающей на пол персоны, фон её прямоугольника имеет цвет, зависящий от пола.

### Описание графических компонентов формы DeletingPerson

Форма DeletingPerson – это вспомогательная форма, которая даёт пользователю возможность отменить удаление текущей персоны и при удалении родителя-одиночки выбрать вариант манипуляций над его детьми.

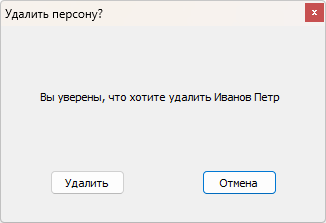


Рисунок 17 – Предупреждение об удалении персоны

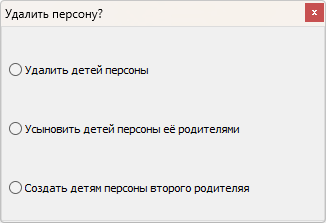


Рисунок 18 – Варианты манипуляций над детьми персоны

### Описание графических компонентов формы Help

На данной форме расположен один компонент – TwebBrowser, который отображает информацию о программном средстве.

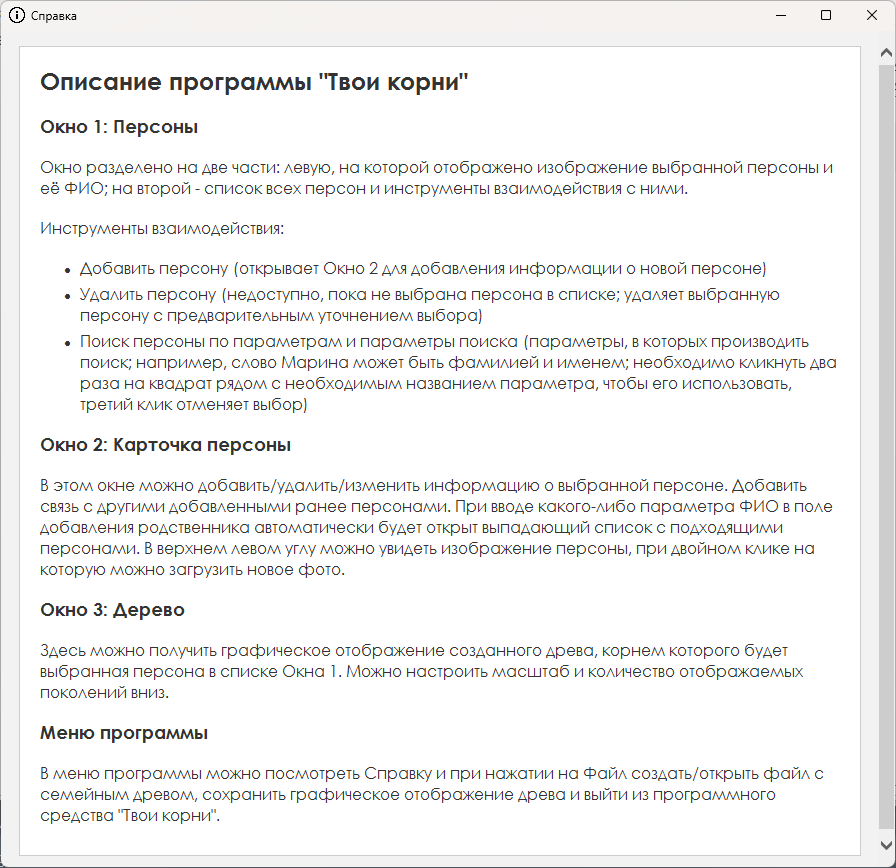


Рисунок 19 - Справка программного средства «Твои корни»

# Тестирование и проверка работоспособности программного средства

## Запуск программы

### Тест 1

Таблица 3.1 – Тест 1

|  |  |
| --- | --- |
| Тестовая  ситуация: | Проверка корректности поведения программы при запуске программы |
| Исходный набор данных: | Запуск программы |
| Ожидаемый результат: | Открытие главной формы |
| Фактический результат: |  |

## Работа с персонами

### Тест 2

Таблица 3.2 – Тест 2

|  |  |
| --- | --- |
| Тестовая  ситуация: | Проверка возможности добавления персоны |
| Исходный набор данных: | Нажатие на кнопку «Добавить», изменение данных, преход на Вкладку 1 |
| Ожидаемый результат: | Открытие Вкладки 2, возможность добавления информации, появление в списке новой персоны |

Продолжение Таблицы 3.2

|  |  |
| --- | --- |
| Фактический результат: |  |

### Тест 3

Таблица 3.3 – Тест 3

|  |  |
| --- | --- |
| Тестовая  ситуация: | Проверка корректности поиска |
| Исходный набор данных: | Ввод в поиск в списке персон (Жук Яна, Иванов Петр, Жук Петр) «Ж» и нажатие на параметр «Фамилия» |
| Ожидаемый результат: | Показ списка персон Жук Яна и Жук Петр |

Продолжение Таблицы 3.3

|  |  |
| --- | --- |
| Фактический результат: |  |

### Тест 4

Таблица 3.4 – Тест 4

|  |  |
| --- | --- |
| Тестовая  ситуация: | Проверка раскрытия выпадающего списка и автозаполнения при вводе родственника |
| Исходный набор данных: | Список персон (Жук Яна), ввод «Ж» в TComboBox |
| Ожидаемый результат: | Раскрытие выпадающего списка и автозаполнение при вводе родственника |
| Фактический результат: |  |

### Тест 5

Таблица 3.5 – Тест 5

|  |  |
| --- | --- |
| Тестовая  ситуация: | Проверка отображения нескольких детей |
| Исходный набор данных: | Список детей (Ребенок 1, Ребенок 2) у персоны |
| Ожидаемый результат: | Отображение всех детей в TListBox |

Продолжение Таблицы 3.5

|  |  |
| --- | --- |
| Фактический результат: |  |

## Отрисовка дерева

### Тест 6

Таблица 3.6 – Тест 6

|  |  |
| --- | --- |
| Тестовая  ситуация: | Проверка отрисовки 1-го поколения в двух масштабах |
| Исходный набор данных: | Список персон (Жук Яна, Иванов Петр, Жук Петр), нажатие на 100% и на 50% масштаба |
| Ожидаемый результат: | Отрисовка 1-го поколения сперва в 100% масштабе, после – в 50% |
| Фактический результат: |  |

### Тест 7

Таблица 3.7 – Тест 7

|  |  |
| --- | --- |
| Тестовая  ситуация: | Проверка отрисовки 2-го поколения |
| Исходный набор данных: | Список персон (Жук Яна, Иванов Петр, Жук Петр), нажатие на 100% и на значения 2-го поколения |
| Ожидаемый результат: | Отрисовка 2-го поколения в 100% масштабе |
| Фактический результат: |  |

## Экспорт дерева

### Тест 8

Таблица 3.8 – Тест 8

|  |  |
| --- | --- |
| Тестовая  ситуация: | Проверка экспорта дерева в PNG |
| Исходный набор данных: | Графическое предствление дерева в программном средстве |
| Ожидаемый результат: | Сохраненный файл с расширением PNG |
| Фактический результат: |  |

## Создание и открытие типизированных файлов

### Тест 9

Таблица 3.9 – Тест 9

|  |  |
| --- | --- |
| Тестовая  ситуация: | Проверка создания файла |
| Исходный набор данных: | Нажатие на кнопку меню «Создать» |
| Ожидаемый результат: | Сохраненный файл с расширением GED |
| Фактический результат: |  |

### Тест 10

Таблица 3.10 – Тест 10

|  |  |
| --- | --- |
| Тестовая  ситуация: | Проверка открытия файла GED |
| Исходный набор данных: | Нажатие на кнопку меню «Открыть» и выбор файла с расширением GED |
| Ожидаемый результат: | Заполненный список из файла GED |
| Фактический результат: |  |

# Руководство по установке

## Минимальные системные требования

Для успешного запуска данного программного средства и комфортной работы с ним необходимо соответствие минимальным системным требованиям:

– процессор 1000 МГц или выше;

– объем оперативной памяти не менее 512 МБ;

– свободное место на диске не менее 20.0 МБ;

– операционная система Windows 7 и выше.

## Установка

На установочном диске находится установочный файл с программным средством. После открытия данного пакета на экране появляется окно, представленное на рисунке 20. В данном окне у пользователя есть возможность выбрать путь для установки, а также пользователь может заметить, что для загрузки приложения на диск требуется как минимум 20.0 МБ свободного дискового пространства для продолжения установки программного средства.

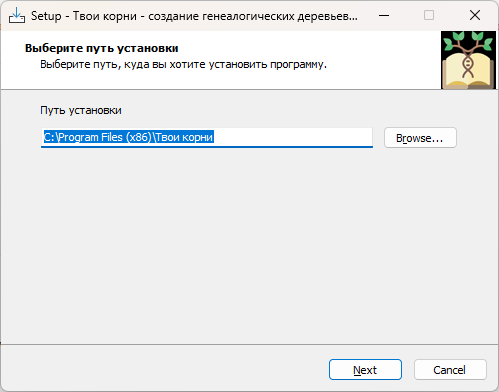


Рисунок 20 – Установка (Этап 1)

После этого этапа у пользователя есть возможность выбрать дополнительные параметры загрузки. У установщика данного программного средства этой задачей является добавление значка на Рабочий стол. Данное Окно имеет вид, представленный на рисунке 21.

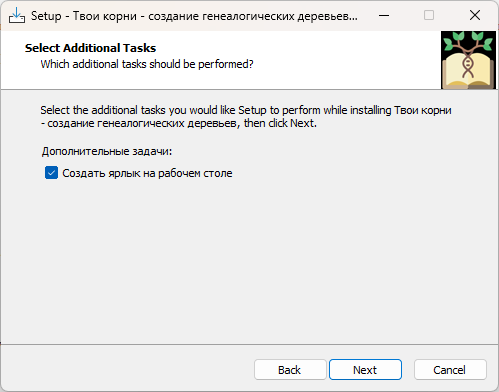


Рисунок 21 – Установка (Этап 2)

После этого этапа подготовка к установке завершается и пользователю отображается предложение установить программное средство на компьютер, показанное на рисунке 22.

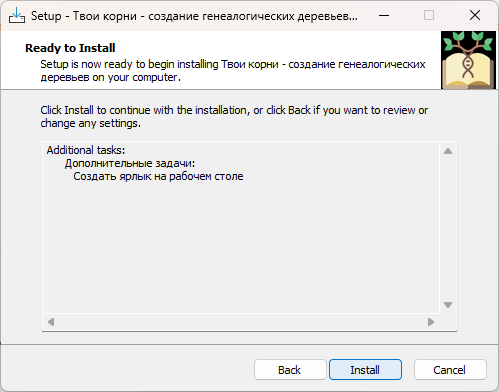


Рисунок 22 -Установка (Этап 3)

Об успешном завершении установки пользователя информирует окно, представленное на рисунке 23.

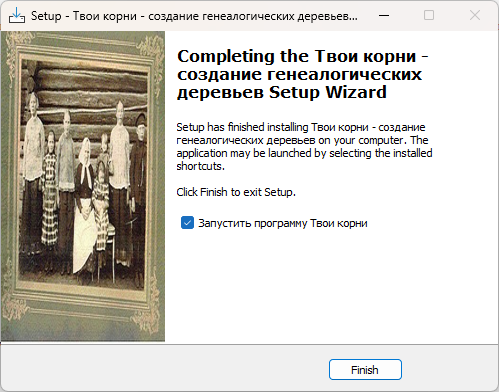


Рисунок 23 – Установка (Этап 4)

## Работа с приложением

После установки и открытия программного средства «Твои корни» на экране появляется окно для работы, представленное на рисунке 24.

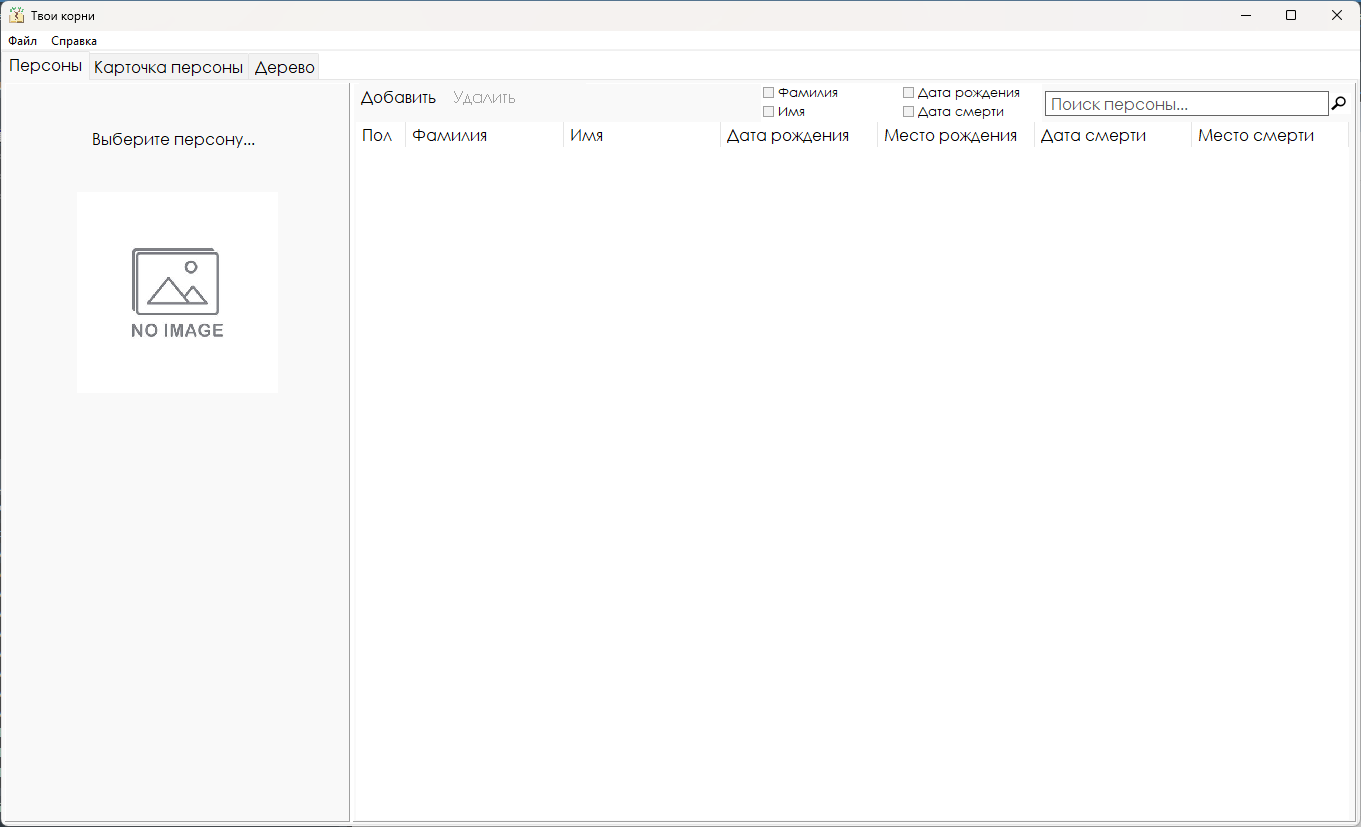


Рисунок 24 – Начальный экран

Вкладка 1: Персоны. Окно разделено на две части: левую, на которой отображено изображение выбранной персоны и её ФИО; на второй – список всех персон и инструменты взаимодействия с ними:

* добавить персону (открывает Вкладку 2 для добавления информации о новой персоне);
* удалить персону (недоступно, пока не выбрана персона в списке; удаляет выбранную персону с предварительным уточнением выбора, если персона – родитель-одиночка);
* поиск персоны по параметрам и параметры поиска (параметры, в которых производить поиск; например, слово Марина может быть фамилией и именем; необходимо кликнуть два раза на квадрат рядом с необходимым названием параметра, чтобы его использовать, третий клик отменяет выбор).

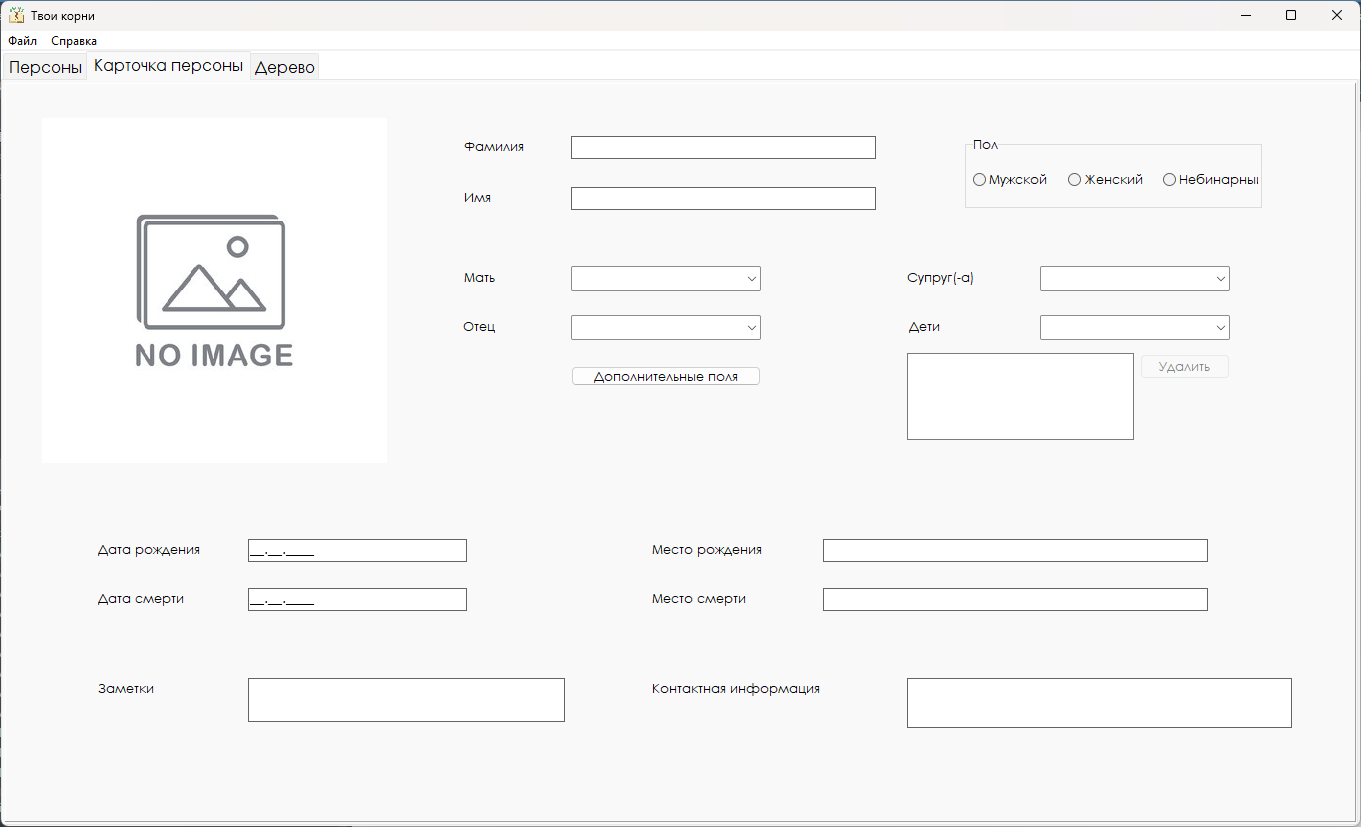


Рисунок 25 – Вторая вкладка компонента PageControl

Вкладка 2: Карточка персоны. В этом окне можно добавить/удалить/изменить информацию о выбранной персоне. Добавить связь с добавленными ранее персонами с помощью различных компонентов, которые являются выпадающими списками и при вводе какого-либо параметра ФИО в поле добавления родственника автоматически будет открыт выпадающих список с подходящими персонами. Для удобства добавление детей происходит также, однако их отображение имеет другой вид: все дети персоны будут отображены в небольшом списке, который удобен в данной ситуации. Также рядом расположена кнопка удаления, которая станет активной после того, как будет выбран нужный ребенок. Для выбора пола персоны в правом верхнем углу находится группа радио кнопок, что позволяет выбрать только лишь одно значение пола. В верхнем левом угле отображено изображение персоны, при двойном клике на которую можно загрузить новое фото. При удалении родителя-одиночки решением может стать усыновление его детей вместо их удаления. В таком случае у усыновленных персон будут приемные родители, поменять/присвоить которых можно при нажатии на кнопку «Дополнительные поле».

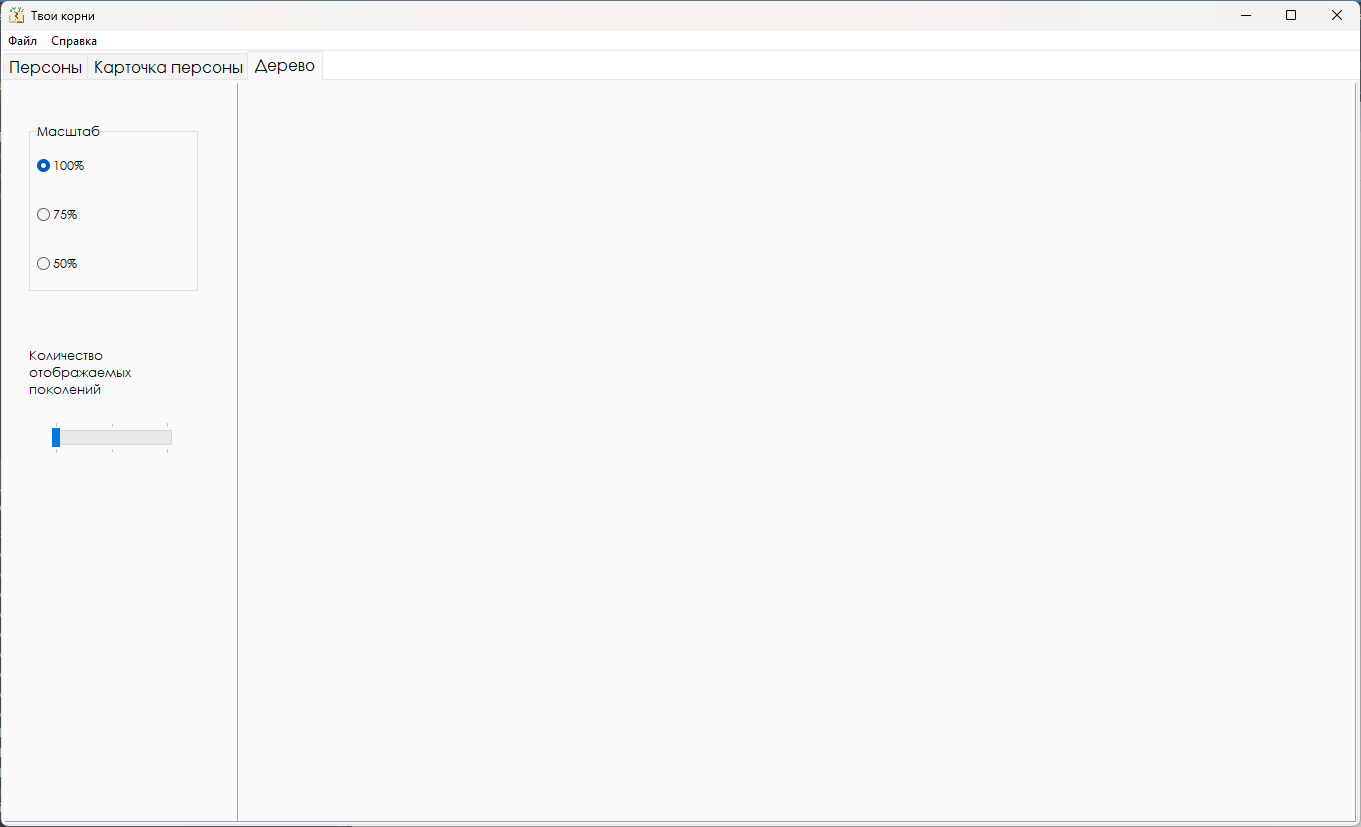


Рисунок – Третья вкладка компонента PageControl

Вкладка 3: Дерево. Здесь можно получить графическое отображение созданного древа, корнем которого будет выбранная персона в списке Вкладка 1. Есть возможность настроить масштаб и количество отображаемых поколений вниз относительно корня древа. Кроме того, при отображении вместо подписи, указывающей на пол персоны, фон её прямоугольника имеет цвет, зависящий от пола.

Заключение

По итогу работы над курсовым проектом было разработано программное средство "Создание генеалогических деревьев", обладающее графическим интерфейсом для взаимодействия с пользователем.

Приложение прошло все этапы тестирования и продемонстрировало корректную и удобную работу, обработав исключительные ситуации и предоставив возможность создавать генеалогические деревья. Также был проверен вспомогательный функционал, позволяющий сохранять и загружать созданные деревья, добавлять, удалять и редактировать данные о родственниках.

Данное программное средство обладает всеми необходимыми функциями для создания и визуализации генеалогических деревьев. Использование приложения в повседневной жизни позволит легко и удобно вести генеалогические исследования, сохранять информацию о родственниках и визуализировать их связи.

Бесценным результатом курсового проектирования является полученный опыт работы с графическим интерфейсом пользователя (изучение компонентов форм, свойств и методов объектов, событий), а также с работой с данными, сохранением и загрузкой из файлов. Был получен опыт работы с динамическими структурами данных, обобщены и применены все ранее полученные знания.

В перспективе данное программное средство может быть усовершенствовано (оптимизация кода, добавление новых функций, улучшение визуального оформления и т. д.), чтобы обеспечить еще большую удобство и функциональность при работе с генеалогическими данными.

Список литературы

[1] Кнут Д.Э. Искусство программирования: Учеб. пособие. Т. 1. Основные алгоритмы. – М.: Вильямс, 2000. – 722 с.: ил.

[2] Вирт, Н. Алгоритмы и структуры данных. – М.: Мир, 1989. – 360 с.: ил.

[3] Серебряная, Л.В. Электронный учебно-методический комплекс по дисциплине «Структуры и алгоритмы обработки данных» [Электронный ресурс]. Режим доступа: <https://erud.bsuir.by/>. – Дата доступа: 15.03.2023.

[4] Серебряная, Л.В. Структуры и алгоритмы обработки данных : учеб.-метод. Пособие / Л. В. Серебряная, И. М. Марина. – Минск : БГУИР, 2013 – 51 с.

[5] Фленов, М. Е. Библия Delphi. – 3-е изд., перераб. и доп. – СПб.: БХВ-Петербург, 2011. – 688 с.: ил.

[6] Глухова, Л. А. Электронный учебно-методический комплекс по дисциплине «Основы алгоритмизации и программирования». Часть 1. [Электронный ресурс]. Режим доступа: <https://erud.bsuir.by/>. – Дата доступа: 15.03.2023.

[7] The GEDCOM Standard, Release 5.5.1. [Электронный ресурс]. - Режим доступа: https://gedcom.io/specifications/ged551.pdf – Дата доступа: 15.03.2023.

[8] Объектно-ориентированное программирование на языках Delphi и C++ : учебное пособие для студентов [Электронный ресурс] / А. Н. Вальвачев, К. А. Сурков, Д. А. Сурков, Ю. М. Четырько. – БГУИР, 2016 . – 432 с.

Приложение А

(обязательное)

Исходный код программы (модуль Work)

interface

uses

Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants, System.Classes, Vcl.Graphics, uManipulWithPerson, uForFiles, uhelp, Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.ComCtrls, Vcl.Menus, Vcl.ExtCtrls, Vcl.ToolWin, Vcl.StdCtrls, Vcl.WinXCtrls, Vcl.Mask, System.Actions, Vcl.ActnList, Jpeg, Vcl.ExtDlgs, uDeletePerson, Vcl.CheckLst;

type

ArrOfPersons = array of PPerson;

TFwork = class(TForm)

MainMenu: TMainMenu;

N1: TMenuItem;

N2: TMenuItem;

N3: TMenuItem;

N4: TMenuItem;

N6: TMenuItem;

N7: TMenuItem;

N9: TMenuItem;

N11: TMenuItem;

PageControl: TPageControl;

TSPersons: TTabSheet;

TSTree: TTabSheet;

TSPersonCard: TTabSheet;

PListOfPersons: TPanel;

SBPersons: TSearchBox;

TBForList: TToolBar;

LVPersons: TListView;

PPersonInfoShort: TPanel;

Splitter1: TSplitter;

IPersonShort: TImage;

LParentsShort: TLabel;

LSpouseShort: TLabel;

LChildrenShort: TLabel;

PPersonInfo: TPanel;

IPersonFull: TImage;

LLastName: TLabel;

LFirstName: TLabel;

TBtnAddPerson: TToolButton;

TBtnDelete: TToolButton;

LMother: TLabel;

LSpousesFull: TLabel;

LBirthDate: TLabel;

LDeathDate: TLabel;

LBirthPlace: TLabel;

LDeathPlace: TLabel;

LNotes: TLabel;

LContactInfo: TLabel;

MELastName: TMaskEdit;

MEFirstName: TMaskEdit;

CBMother: TComboBox;

CBSpouse: TComboBox;

CBFather: TComboBox;

LFather: TLabel;

LStepMother: TLabel;

LStepFather: TLabel;

CBStepMother: TComboBox;

CBStepFather: TComboBox;

LChildrens: TLabel;

CBChild: TComboBox;

BAdditional: TButton;

MEBirthDate: TMaskEdit;

MEDeathDate: TMaskEdit;

MEBirthPlace: TMaskEdit;

MEDeathPlace: TMaskEdit;

MNotes: TMemo;

MContactInfo: TMemo;

LFullName: TLabel;

PTreeSettings: TPanel;

PTreeView: TPanel;

PBTree: TPaintBox;

ActionList: TActionList;

ActOpenPersonCard: TAction;

TrckBGenerations: TTrackBar;

LGenerations: TLabel;

ActDrawTree: TAction;

RGScale: TRadioGroup;

RGSex: TRadioGroup;

ActLoadPicture: TAction;

LstBChildren: TListBox;

BDeleteChild: TButton;

LVsearch: TListView;

ChckL: TCheckListBox;

procedure PageControlChange(Sender: TObject);

// procedure LVPersonsColumnClick(Sender: TObject; Column: TListColumn);

procedure ActDrawTreeExecute(Sender: TObject);

procedure ActOpenPersonCardExecute(Sender: TObject);

procedure FormActivate(Sender: TObject);

procedure AddToTheList(const Person: PPerson; NeededList: TListView = nil);

procedure ShowInfoInCard(Person: PPerson = nil);

procedure AddToTheCombos(Person: PPerson);

function AmountOfBlocks(var WHeight: word; WWidth, CWidth, CHeight: word;

Margins: byte): byte;

procedure PageControlChanging(Sender: TObject; var AllowChange: Boolean);

procedure LVPersonsSelectItem(Sender: TObject; Item: TListItem;

Selected: Boolean);

procedure BAdditionalClick(Sender: TObject);

procedure ActLoadPictureExecute(Sender: TObject);

procedure LstBChildrenClick(Sender: TObject);

procedure CBChildSelect(Sender: TObject);

procedure ChangeValueInCombos(Value: PPerson; Delete: Boolean = False);

procedure TBtnDeleteClick(Sender: TObject);

procedure SBPersonsFind(Sender: TObject);

procedure N9Click(Sender: TObject);

procedure N2Click(Sender: TObject);

procedure N3Click(Sender: TObject);

procedure FormClose(Sender: TObject; var Action: TCloseAction);

procedure N11Click(Sender: TObject);

procedure N6Click(Sender: TObject);

end;

var

Fwork: TFwork;

implementation

var

scale: real = 1;

CurrentPerson: PPerson = nil;

NewPerson: Boolean = False;

CurrentChild: PPerson = nil;

CurrPIndex: integer;

PathForFileS, PathForFileR: string;

nopicture: string = 'D:\Универ\Курсач\Иконки\No Picture.jpg';

PathForPictures: string = 'D:\Универ\Курсач\Картинки\';

PathForFile: string = 'D:\Универ\Курсач\Файлики';

{$R \*.dfm}

procedure ResetControls(AContainer: TWinControl);

var

i: integer;

begin

for i := 0 to AContainer.ControlCount - 1 do

begin

if AContainer.Controls[i] is TComboBox then

TCustomComboBox(AContainer.Controls[i]).ItemIndex := -1

else if AContainer.Controls[i] is TRadioGroup then

TRadioGroup(AContainer.Controls[i]).ItemIndex := -1

Else if AContainer.Controls[i] is TMaskEdit then

TMaskEdit(AContainer.Controls[i]).Text := ''

Else if AContainer.Controls[i] is TListBox then

TListBox(AContainer.Controls[i]).Clear;

end;

end;

procedure TFwork.ChangeValueInCombos(Value: PPerson; Delete: Boolean = False);

var

i, j, IndexToDelete: integer;

AllCB: array [1 .. 4] of TComboBox;

begin

AllCB[1] := CBFather;

AllCB[2] := CBMother;

AllCB[3] := CBChild;

AllCB[4] := CBSpouse;

for i := 1 to 4 do

begin

j := 0;

IndexToDelete := -1;

repeat

if PPerson(AllCB[i].Items.Objects[j]) = Value then

IndexToDelete := j;

inc(j);

until (IndexToDelete <> -1) or (j > AllCB[i].Items.Count - 1);

if IndexToDelete <> -1 then

Begin

AllCB[i].Items[IndexToDelete] := GetFullname(Value);

if Delete then

AllCB[i].Items.Delete(IndexToDelete);

End;

end;

end;

function TFwork.AmountOfBlocks(var WHeight: word; WWidth, CWidth, CHeight: word;

Margins: byte): byte;

Begin

CWidth := CWidth + Margins;

WHeight := WHeight - CHeight - Margins;

Result := WWidth div CWidth;

End;

function AmountOfGen(Gen: byte; Root: PPerson; var Persons: ArrOfPersons;

index: byte = 0): word;

var

CurrentSubPerson: PPerson;

Begin

Result := 0;

case Gen of

2:

begin

if Root^.Children <> nil then

Begin

CurrentSubPerson := Root^.Children;

repeat

Persons[index] := CurrentSubPerson;

inc(index);

inc(Result);

CurrentSubPerson := CurrentSubPerson^.Sibling;

until (CurrentSubPerson = nil) or (index = length(Persons));

Persons[index] := nil;

inc(index);

End

else

Persons[index] := nil;

end;

3:

begin

if Root^.Children^.Children <> nil then

Begin

Result := AmountOfGen(2, Root, Persons, index);

CurrentSubPerson := Root^.Children;

index := 0;

While (CurrentSubPerson <> nil) and (index < length(Persons)) do

begin

Result := Result + AmountOfGen(Gen - 1, CurrentSubPerson,

Persons, index);

CurrentSubPerson := CurrentSubPerson^.Sibling;

end;

End;

end;

end;

End;

procedure TFwork.BAdditionalClick(Sender: TObject);

begin

TButton(Sender).Visible := False;

LStepMother.Visible := True;

CBStepMother.Visible := True;

LStepFather.Visible := True;

CBStepFather.Visible := True;

end;

procedure TFwork.CBChildSelect(Sender: TObject);

var

IndexOfP: integer;

Person: PPerson;

begin

IndexOfP := TComboBox(Sender).ItemIndex;

if IndexOfP <> -1 then

Begin

Person := PPerson(TComboBox(Sender).Items.Objects[IndexOfP]);

LstBChildren.Items.AddObject(GetFullname(Person), TObject(Person));

AddPerson(CurrentPerson, Person, 'Children');

End;

end;

procedure TFwork.ActDrawTreeExecute(Sender: TObject);

var

WH, WW, dx, dy: word;

PersonCard, PCWork, Names, ImageR, Dates, WorkArea: TRect;

PCHeight, PCWidth, PCWHeight, PCWWidth: word;

NRectLeft, NRectBottom, i: word;

Generation, PersonInGen, CurrPersonInGen: byte;

ForOutput: string;

ImageOfP: TPicture;

ScaleGen: real;

Margins: byte;

NextPersons: ArrOfPersons;

begin

WH := PBTree.ClientHeight;

WW := PBTree.ClientWidth;

dx := WW div 2;

dy := WH div 2;

PBTree.Canvas.Brush.Color := RGB(230, 230, 250);

PBTree.Canvas.FillRect(PBTree.ClientRect);

Margins := WW div 60;

PBTree.Canvas.Pen.Color := cLBlack;

PBTree.Canvas.Pen.Width := 2;

Generation := 1;

ImageOfP := TPicture.Create;

if Sender is TRadioGroup then

case TRadioGroup(Sender).ItemIndex of

0:

scale := 1;

1:

scale := 0.75;

2:

scale := 0.5;

end;

if Sender is TTrackBar then

Generation := TTrackBar(Sender).Position;

PersonCard.Create(7 \* (WW div 18), (WH div 2) - (WH div 10), 11 \* (WW div 18),

(WH div 2) + (WH div 10));

PCHeight := PersonCard.Bottom - PersonCard.Top;

PCWidth := PersonCard.Right - PersonCard.Left;

SetLength(NextPersons, 1);

NextPersons[0] := CurrentPerson;

for i := 1 to Generation do

Begin

case i of

1:

begin

ScaleGen := scale \* 1.2;

if Assigned(CurrentPerson^.Spouse) then

Begin

SetLength(NextPersons, 2);

NextPersons[1] := CurrentPerson^.Spouse;

PersonInGen := 2;

End

else

PersonInGen := 1;

end;

2:

begin

SetLength(NextPersons, AmountOfBlocks(WH, WW, PCWidth, PCHeight,

Margins) + 2);

ScaleGen := scale;

PersonInGen := AmountOfGen(2, CurrentPerson, NextPersons);

end;

3:

begin

ScaleGen := scale \* 0.8;

SetLength(NextPersons, AmountOfBlocks(WH, WW,

round(ScaleGen \* PCWidth), round(ScaleGen \* PCHeight),

Margins) + 2);

PersonInGen := AmountOfGen(3, CurrentPerson, NextPersons);

end;

end;

CurrPersonInGen := 1;

dy := 0;

dx := 0;

Repeat

WorkArea.Right := (WW div PersonInGen) \* (CurrPersonInGen);

WorkArea.Left := (WW div PersonInGen) \* (CurrPersonInGen - 1);

WorkArea.Top := (i - 1) \* (WH div Generation);

WorkArea.Bottom := i \* (WH div Generation);

dx := (WorkArea.Bottom + WorkArea.Top) div 2;

dy := (WorkArea.Right + WorkArea.Left) div 2;

PersonCard.Right := round(dy + (PCWidth div 2) \* ScaleGen);

PersonCard.Left := round(dy - (PCWidth div 2) \* ScaleGen);

PersonCard.Top := round(dx - (PCHeight div 2) \* ScaleGen);

PersonCard.Bottom := round(dx + (PCHeight div 2) \* ScaleGen);

ImageOfP.LoadFromFile(NextPersons[CurrPersonInGen - 1]^.Photo);

// НЕ МЕНЯТЬ, ЗАВИСИМОСТЬ ОТ ОСНОВНОЙ КАРТОЧКИ

PCWork.Create(PersonCard.Left + 10, PersonCard.Top + 10,

PersonCard.Right - 10, PersonCard.Bottom - 10);

PCWHeight := PCWork.Bottom - PCWork.Top;

PCWWidth := PCWork.Right - PCWork.Left;

NRectLeft := round(PCWork.Left + 1.5 \* (PCWWidth div 3));

NRectBottom := PCWork.Bottom - (PCWHeight div 3);

Names.Create(NRectLeft, PCWork.Top, PCWork.Right, NRectBottom);

inflaterect(Names, -3, -3);

ImageR.Create(PCWork.Left, PCWork.Top, Names.Left, Names.Bottom);

inflaterect(ImageR, -7, -3);

Dates.Create(PCWork.Left, Names.Bottom + 5, PCWork.Right,

PCWork.Bottom - 5);

inflaterect(Dates, -3, -3);

case NextPersons[CurrPersonInGen - 1]^.Gender of

Male:

PBTree.Canvas.Brush.Color := RGB(178, 223, 238);

Female:

PBTree.Canvas.Brush.Color := RGB(255, 204, 204);

NonBinary:

PBTree.Canvas.Brush.Color := RGB(211, 211, 211);

end;

PBTree.Canvas.RoundRect(PersonCard, 20, 20);

PBTree.Canvas.Font.Size := round(11 \* ScaleGen);

PBTree.Canvas.Font.Name := 'Century Gothic';

ForOutput := GetFullname(NextPersons[CurrPersonInGen - 1]);

// рассчитываем размеры текста и выводим его внутри прямоугольника

PBTree.Canvas.TextRect(Names, ForOutput, [tfCenter, tfWordBreak]);

ForOutput := '';

if NextPersons[CurrPersonInGen - 1]^.DOfBirth <> 0 then

Begin

ForOutput := DateTostr(NextPersons[CurrPersonInGen - 1]^.DOfBirth);

if not NextPersons[CurrPersonInGen - 1]^.Alive then

ForOutput := ForOutput + ' - ' +

DateTostr(NextPersons[CurrPersonInGen - 1]^.DOFDEath);

End;

PBTree.Canvas.TextRect(Dates, ForOutput, [tfCenter, tfWordBreak]);

PBTree.Canvas.StretchDraw(ImageR, ImageOfP.Graphic);

inc(CurrPersonInGen);

Until (NextPersons[CurrPersonInGen - 1] = nil) or

(CurrPersonInGen > PersonInGen);

End;

end;

procedure TFwork.ShowInfoInCard(Person: PPerson = nil);

var

Child: PPerson;

begin

if Person <> nil then

Begin

CurrentPerson := Person;

if Person^.DOfBirth <> 0 then

MEBirthDate.Text := DateTimeToStr(Person^.DOfBirth);

MEBirthPlace.Text := Person^.POfbirth.Country + ', ' + Person^.POfbirth.City

+ ', ' + Person^.POfbirth.Address;

if not Person^.Alive then

Begin

if Person^.DOFDEath <> 0 then

MEDeathDate.Text := DateTimeToStr(Person^.DOFDEath);

MEDeathPlace.Text := Person^.POfDeath.Country + ', ' +

Person^.POfDeath.City + ', ' + Person^.POfDeath.Address;

MContactInfo.lines.add(Person^.contactinfo);

End;

MEFirstName.Text := Person^.fname;

MELastName.Text := Person^.lname;

MNotes.lines.add(Person^.contactinfo);

RGSex.ItemIndex := ord(Person^.Gender);

if Assigned(Person^.Spouse) then

CBSpouse.ItemIndex := CBSpouse.Items.IndexOf(GetFullname(Person^.Spouse));

if Assigned(Person^.Father) then

CBFather.ItemIndex := CBFather.Items.IndexOf(GetFullname(Person^.Father));

if Assigned(Person^.Mother) then

CBMother.ItemIndex := CBMother.Items.IndexOf(GetFullname(Person^.Mother));

Child := Person^.Children;

while Child <> nil do

Begin

LstBChildren.Items.AddObject(GetFullname(Child), TObject(Child));

Child := Child^.Sibling;

End;

if CurrentPerson^.Photo <> '' then

IPersonFull.Picture.LoadFromFile(CurrentPerson^.Photo);

End;

end;

procedure TFwork.TBtnDeleteClick(Sender: TObject);

begin

FDeletePerson.DeletePersonShow(CurrentPerson);

TBtnDelete.Enabled := False;

LVPersons.Items[CurrPIndex].Delete;

ChangeValueInCombos(CurrentPerson, True);

if CurrentPerson <> nil then

Begin

AddToTheList(CurrentPerson, LVPersons);

AddToTheCombos(CurrentPerson);

End;

end;

procedure TFwork.AddToTheList(const Person: PPerson;

NeededList: TListView = nil);

var

PersonItem: TListItem;

begin

PersonItem := NeededList.Items.add;

PersonItem.data := Person;

case Person^.Gender of

Male:

PersonItem.caption := 'М';

Female:

PersonItem.caption := 'Ж';

NonBinary:

PersonItem.caption := 'Неб';

end;

PersonItem.SubItems.add(Person^.lname);

PersonItem.SubItems.add(Person^.fname);

if Person^.DOfBirth <> 0 then

PersonItem.SubItems.add(DateTimeToStr(Person^.DOfBirth))

Else

PersonItem.SubItems.add('');

PersonItem.SubItems.add(Person^.POfbirth.Country + ', ' +

Person^.POfbirth.City + ', ' + Person^.POfbirth.Address);

if not Person^.Alive then

Begin

if Person^.DOFDEath <> 0 then

PersonItem.SubItems.add(DateTimeToStr(Person^.DOFDEath))

Else

PersonItem.SubItems.add('');

PersonItem.SubItems.add(Person^.POfDeath.Country + ', ' +

Person^.POfDeath.City + ', ' + Person^.POfDeath.Address);

End;

end;

procedure StartAdd(const Person: PPerson);

begin

Fwork.AddToTheList(Person, Fwork.LVPersons);

Fwork.AddToTheCombos(Person);

end;

procedure TFwork.AddToTheCombos(Person: PPerson);

begin

if (Person^.Gender = Male) or (Person^.Gender = NonBinary) then

CBFather.AddItem(GetFullname(Person), Pointer(Person));

if (Person^.Gender = Female) or (Person^.Gender = NonBinary) then

CBMother.AddItem(GetFullname(Person), Pointer(Person));

CBSpouse.AddItem(GetFullname(Person), Pointer(Person));

CBChild.AddItem(GetFullname(Person), Pointer(Person));

end;

procedure TFwork.ActLoadPictureExecute(Sender: TObject);

var

OpenPictureDialog: TOpenPictureDialog;

FilePath: String;

begin

try

OpenPictureDialog := TOpenPictureDialog.Create(nil);

// Устанавливаем фильтр для диалогового окна

OpenPictureDialog.Filter := 'Изображения|\*.bmp;\*.jpg;\*.png';

// Проверяем, если пользователь выбрал файл и нажал "ОК"

if OpenPictureDialog.Execute then

begin

// Получаем путь к выбранному файлу

FilePath := OpenPictureDialog.FileName;

// Загружаем изображение в TImage

TImage(Sender).Picture.LoadFromFile(OpenPictureDialog.FileName);

if CurrentPerson^.fname <> '' then

CurrentPerson^.Photo := PathForPictures + GetFullname(CurrentPerson) +

Copy(OpenPictureDialog.FileName, Pos('.', OpenPictureDialog.FileName))

else

CurrentPerson^.Photo := PathForPictures +

StringReplace(CurrentPerson^.index, '@', '', [rfReplaceAll]) +

Copy(OpenPictureDialog.FileName,

Pos('.', OpenPictureDialog.FileName));

TImage(Sender).Picture.SaveToFile(CurrentPerson^.Photo);

end;

finally

OpenPictureDialog.Free;

end;

end;

procedure TFwork.ActOpenPersonCardExecute(Sender: TObject);

var

NewP: PPerson;

index: String;

begin

PageControl.ActivePage := TSPersonCard;

if Sender is TToolButton then

Begin

New(NewP);

CurrentPerson := NewP;

NewPerson := True;

Randoming(index);

CurrentPerson^.index := index;

CurrentPerson^.Gender := TGender(2);

CurrentPerson^.Mother := nil;

CurrentPerson^.Father := nil;

CurrentPerson^.StepM := nil;

CurrentPerson^.StepF := nil;

CurrentPerson^.Spouse := nil;

CurrentPerson^.Sibling := nil;

CurrentPerson^.Children := nil;

End;

end;

procedure TFwork.FormActivate(Sender: TObject);

begin

IPersonShort.Picture.LoadFromFile(nopicture);

IPersonFull.Picture.LoadFromFile(nopicture);

PathForPictures := ExtractFilePath(Application.ExeName);

end;

procedure TFwork.FormClose(Sender: TObject; var Action: TCloseAction);

begin

if PathForFileS <> '' then

WriteIntoFile(CurrentPerson, PathForFileS);

end;

procedure TFwork.LstBChildrenClick(Sender: TObject);

begin

if TListBox(Sender).ItemIndex <> -1 then

begin

CurrentChild := PPerson(TListBox(Sender).Items.Objects

[TListBox(Sender).ItemIndex]);

BDeleteChild.Enabled := True;

end

Else

begin

CurrentChild := nil;

BDeleteChild.Enabled := False;

end;

end;

procedure TFwork.LVPersonsSelectItem(Sender: TObject; Item: TListItem;

Selected: Boolean);

begin

if Selected then

begin

CurrentPerson := Item.data;

CurrPIndex := Item.index;

LFullName.caption := GetFullname(CurrentPerson);

TBtnDelete.Enabled := True;

if CurrentPerson^.Photo <> '' then

IPersonShort.Picture.LoadFromFile(CurrentPerson^.Photo);

end;

end;

procedure TFwork.N3Click(Sender: TObject);

Var

OpenDialog: TOpenDialog;

begin

OpenDialog := TOpenDialog.Create(Self);

try

OpenDialog.Filter := 'Типизированный файл (\*.ged)|\*.ged';

if OpenDialog.Execute then

// Действия при выборе файла

PathForFileR := OpenDialog.FileName;

finally

OpenDialog.Free;

end;

CurrentPerson := ReadPersonsFromFile(PathForFileR);

visit := TStringList.Create;

GoingThroughTree(CurrentPerson, visit, [@StartAdd]);

visit.Free;

end;

procedure TFwork.N6Click(Sender: TObject);

var

SaveDialog: TSaveDialog;

Image: TBitmap;

JpegImage: TJpegImage;

begin

SaveDialog := TSaveDialog.Create(nil);

try

SaveDialog.Title := 'Сохранение изображения семейного древа';

// Set the file filters for different formats

SaveDialog.Filter :=

'JPEG Image (\*.jpg)|\*.jpg|PNG Image (\*.png)|\*.png|All Files (\*.\*)|\*.\*';

if SaveDialog.Execute then

begin

if SameText(ExtractFileExt(SaveDialog.FileName), '.jpg') then

begin

// Save the image in JPEG format

JpegImage := TJpegImage.Create;

try

JpegImage.Assign(PBTree.Canvas);

JpegImage.SaveToFile(SaveDialog.FileName);

finally

JpegImage.Free;

end;

end

else if SameText(ExtractFileExt(SaveDialog.FileName), '.png') then

begin

// Save the image in PNG format

Image := TBitmap.Create;

try

Image.Assign(PBTree.Canvas);

Image.SaveToFile(SaveDialog.FileName);

finally

Image.Free;

end;

end;

end;

finally

SaveDialog.Free;

end;

end;

procedure TFwork.N11Click(Sender: TObject);

begin

FHelp.Execute('Info');

end;

procedure TFwork.N2Click(Sender: TObject);

var

SaveDialog: TSaveDialog;

begin

SaveDialog := TSaveDialog.Create(Self);

try

SaveDialog.Filter := 'Типизированный файл (\*.ged)|\*.ged';

if SaveDialog.Execute then

begin

// Действия при выборе имени файла

PathForFileS := SaveDialog.FileName + '.ged';

end;

finally

SaveDialog.Free;

end;

end;

procedure TFwork.N9Click(Sender: TObject);

begin

if PathForFileS <> '' then

WriteIntoFile(CurrentPerson, PathForFileS);

Close;

end;

procedure TFwork.PageControlChange(Sender: TObject);

begin

case PageControl.ActivePageIndex of

1:

begin

ShowInfoInCard(CurrentPerson);

end;

2:

begin

if CurrentPerson <> nil then

ActDrawTreeExecute(nil);

end;

end;

end;

procedure RenamePhotoPath(NewString: string; var Path: string);

var

StartIndex, FinishIndex: integer;

Pict: File;

Begin

AssignFile(Pict, Path);

StartIndex := length(PathForPictures) + 1;

FinishIndex := Pos('.', Path);

Delete(Path, StartIndex, FinishIndex - StartIndex);

Insert(NewString, Path, StartIndex);

Rename(Pict, Path);

End;

procedure TFwork.PageControlChanging(Sender: TObject; var AllowChange: Boolean);

var

Position: byte;

Change: Boolean;

begin

If PageControl.ActivePageIndex = 1 then

begin

Change := False;

if CurrentPerson <> nil then

begin

if not TryStrToDate(MEBirthDate.Text,

CurrentPerson^.DOfBirth) or

(MEBirthDate.Text = '00.00.0000') then

CurrentPerson^.DOfBirth := 0;

CurrentPerson^.Alive := True;

Position := Pos(', ', MEBirthPlace.Text) - 1;

CurrentPerson^.POfbirth.Country :=

Copy(MEBirthPlace.Text, 1, Position);

CurrentPerson^.POfbirth.City :=

Copy(MEBirthPlace.Text, Position + 3,

Pos(', ', MEBirthPlace.Text, Position) - 1);

Position := Pos(', ', MEBirthPlace.Text, Position) - 1;

CurrentPerson^.POfbirth.Address :=

Copy(MEBirthPlace.Text, Position + 3);

if not TryStrToDate(MEDeathDate.Text,

CurrentPerson^.DOFDEath) or

(MEDeathDate.Text = '00.00.0000') then

CurrentPerson^.DOFDEath := 0;

if CurrentPerson^.DOFDEath <> 0 then

Begin

CurrentPerson^.Alive := False;

Position := Pos(', ', MEDeathPlace.Text) - 1;

CurrentPerson^.POfDeath.Country :=

Copy(MEDeathPlace.Text, 1, Position);

CurrentPerson^.POfDeath.City :=

Copy(MEDeathPlace.Text, Position + 3,

Pos(', ', MEDeathPlace.Text, Position) - 1);

Position := Pos(', ', MEDeathPlace.Text, Position) –

1;

CurrentPerson^.POfDeath.Address :=

Copy(MEDeathPlace.Text, Position + 3);

CurrentPerson^.contactinfo := MContactInfo.Text;

End;

if (CurrentPerson^.fname <> MEFirstName.Text) and not NewPerson then

Change := True;

CurrentPerson^.fname := MEFirstName.Text;

if (CurrentPerson^.lname <> MELastName.Text) and not NewPerson then

Change := True;

CurrentPerson^.lname := MELastName.Text;

CurrentPerson^.contactinfo := MNotes.Text;

if (CurrentPerson^.Gender <> TGender(RGSex.ItemIndex)) and not NewPerson

then

case TGender(RGSex.ItemIndex) of

Male:

LVPersons.Items[CurrPIndex].caption := 'М';

Female:

LVPersons.Items[CurrPIndex].caption := 'Ж';

NonBinary:

LVPersons.Items[CurrPIndex].caption := 'Неб';

end;

CurrentPerson^.Gender := TGender(RGSex.ItemIndex);

if (CBFather.ItemIndex <> -1) and

(PPerson(CBFather.Items.Objects[CBFather.ItemIndex]) <> CurrentPerson)

then

AddPerson(CurrentPerson,

PPerson(CBFather.Items.Objects[CBFather.ItemIndex]), 'Father');

if (CBMother.ItemIndex <> -1) and

(PPerson(CBMother.Items.Objects[CBMother.ItemIndex]) <> CurrentPerson)

then

AddPerson(CurrentPerson,

PPerson(CBMother.Items.Objects[CBMother.ItemIndex]), 'Mother');

if (CBSpouse.ItemIndex <> -1) and

(PPerson(CBSpouse.Items.Objects[CBSpouse.ItemIndex]) <> CurrentPerson)

then

AddPerson(CurrentPerson,

PPerson(CBSpouse.Items.Objects[CBSpouse.ItemIndex]), 'Spouse');

if (CBStepMother.ItemIndex <> -1) and

(PPerson(CBStepMother.Items.Objects[CBMother.ItemIndex]) <>

CurrentPerson) then

begin

CurrentPerson^.StepM :=

PPerson(CBStepMother.Items.Objects[CBStepMother.ItemIndex]);

AddPerson(PPerson(CBStepMother.Items.Objects[CBStepMother.ItemIndex]),

CurrentPerson, 'Children');

end;

if (CBStepFather.ItemIndex <> -1) and

(PPerson(CBStepFather.Items.Objects[CBSpouse.ItemIndex]) <>

CurrentPerson) then

begin

CurrentPerson^.StepF :=

PPerson(CBStepFather.Items.Objects[CBStepFather.ItemIndex]);

AddPerson(PPerson(CBStepFather.Items.Objects[CBStepFather.ItemIndex]),

CurrentPerson, 'Children');

end;

if (CurrentPerson^.Photo = '') or (CurrentPerson^.Photo = nopicture) then

CurrentPerson^.Photo := nopicture

Else

RenamePhotoPath(GetFullname(CurrentPerson), CurrentPerson^.Photo);

if NewPerson then

Begin

AddToTheCombos(CurrentPerson);

AddToTheList(CurrentPerson, LVPersons);

NewPerson := False;

End;

if Change then

begin

ChangeValueInCombos(CurrentPerson);

Change := False;

end;

If CurrPIndex <> -1 then

Begin

LVPersons.Items[CurrPIndex].Delete;

AddToTheList(CurrentPerson, LVPersons);

End;

CurrPIndex := -1;

CurrentPerson := nil;

ResetControls(PPersonInfo);

end;

BAdditional.Visible := True;

LStepMother.Visible := False;

CBStepMother.Visible := False;

LStepFather.Visible := False;

CBStepFather.Visible := False;

IPersonFull.Picture.LoadFromFile(nopicture);

end;

end;

procedure TFwork.SBPersonsFind(Sender: TObject);

var

searchQuery: string;

searchLName, searchFName, searchDOfBirth, searchDOfDeath: Boolean;

Person: PPerson;

i: integer;

begin

searchQuery := SBPersons.Text;

LVsearch.Items.Clear;

if searchQuery = '' then

LVsearch.Visible := False

Else if LVPersons.Items.Count <> 0 then

begin

LVsearch.Visible := True;

searchLName := ChckL.Checked[0];

searchFName := ChckL.Checked[1];

searchDOfBirth := ChckL.Checked[2];

searchDOfDeath := ChckL.Checked[3];

for i := 0 to LVPersons.Items.Count - 1 do

Begin

Person := PPerson(LVPersons.Items[i].data);

If (searchLName and (Pos(searchQuery, Person^.lname) <> 0)) or

(searchFName and (Pos(searchQuery, Person^.fname) <> 0)) or

(searchDOfBirth and (Pos(searchQuery, DateTostr(Person^.DOfBirth)) <> 0)

) or (searchDOfDeath and (Pos(searchQuery, DateTostr(Person^.DOFDEath))

<> 0)) then

AddToTheList(Person, LVsearch);

End;

end;

end;

end.

Приложение Б

(Обязательное)

Исходный код программы (модуль ManipulWithPersons)

unit uManipulWithPerson;

interface

uses classes;

type

TLocation = record

Country, City: String[15];

Address: String[30];

end;

TGender = (Male = 0, Female, Nonbinary);

PPerson = ^TPerson;

TPerson = record

Index: String[9];

Gender: TGender;

Mother, Father, Spouse, StepM, StepF: PPerson;

Children: PPerson;

Adopted: Boolean;

Sibling: PPerson;

POfBirth: TLocation;

FName, LName, Note: String[100];

DOfBirth: TDateTime;

Photo: String;

case Alive: Boolean of

True:

(ContactInfo: String[100]);

False:

(POfDeath: TLocation; DOfDeath: TDateTime);

end;

ProcessNode = procedure(const HeadOf: PPerson);

Processes = array of ProcessNode;

var

Head, RootNode: PPerson;

i: Integer;

AmountOfPersons: byte = 1;

visit: TStringList;

Procedure OutPutInfo(const Person: TPerson; Kind: String = '');

procedure Randoming(var ind: string);

Procedure AddPerson(Root: PPerson; NewPers: PPerson; Role: String = '');

procedure RemovePerson(Person: PPerson; WayOfRemoving: byte = 0);

procedure GoingThroughTree(HeadOfTree: PPerson; var Visited: TStringList;

AOfProcess: Processes = nil);

function GetFullName(Person: PPerson): string;

implementation

uses SysUtils;

{

R D

A O

N M

}

procedure Randoming(var ind: string);

var

i: Integer;

LetterOrNum: byte;

Begin

setlength(ind, 9);

ind[1] := '@';

randomize;

for i := 2 to 8 do

ind[i] := char(ord('0') + Random(9));

ind[9] := '@';

end;

{

A

D

D

}

// Procedure AddPerson(var Root: TPerson; var Amount: byte; Role: String = '');

Procedure AddPerson(Root: PPerson; NewPers: PPerson; Role: String = '');

var

// Relative: String[8];

buf, Child: PPerson;

i: byte;

// Str1, Str2: String[10];

Index: String;

Begin

inc(AmountOfPersons);

// Создаем индекс персоны

Randoming(index);

// Если добавляют мать / отца

if (Role = 'Mother') or (Role = 'Father') then

Begin

if Role = 'Mother' then

Begin

// Если не создана мать

if NewPers = nil then

Begin

New(Root^.Mother);

// Присваиваем мать

NewPers := Root^.Mother;

End

Else

Root^.Mother := NewPers;

// Если почему-то не создан отец

if Root^.Father = nil then

New(Root^.Father);

End

Else if Role = 'Father' then

Begin

// Если не создан отец

if NewPers = nil then

Begin

New(Root^.Father);

// Присваиваем отца

NewPers := Root.Father;

End

Else

Root^.Father := NewPers;

// Если почему-то не создана мать

if Root^.Mother = nil then

New(Root.Mother);

End;

if NewPers^.Children <> nil then

begin

Child := NewPers^.Children;

NewPers^.Children := Root;

Root^.Sibling := Child;

end;

NewPers^.Children := Root;

// Присваивание детям нового родителя

Child := Root^.Sibling;

if Child <> nil then

if Root^.Gender = Female then

while Child <> nil do

Begin

Child^.Father := NewPers;

Child := Child^.Sibling;

End

Else

while Child <> nil do

Begin

Child^.Mother := NewPers;

Child := Child^.Sibling;

End;

End;

if Role = 'Spouse' then

Begin

if NewPers = nil then

Begin

New(Root.Spouse);

NewPers := Root^.Spouse;

End

Else

Root^.Spouse := NewPers;

NewPers^.Spouse := Root;

// Присваивание детям супруга нового родителя

if Root^.Children <> nil then

Begin

Child := Root^.Children;

if Root^.Gender = Female then

while Child <> nil do

Begin

Root^.Children^.Father := NewPers;

Child := Child^.Sibling;

End

Else

while Child <> nil do

Begin

Root^.Children^.Mother := NewPers;

Child := Child^.Sibling;

End;

End

End

Else if Role = 'Children' then

Begin

If Root^.Children = nil then

buf := nil

Else

buf := Root^.Children;

New(Root^.Children);

Root^.Children^.Sibling := buf;

Root^.Children := NewPers;

End;

if NewPers^.Index = '' then

NewPers^.Index := index;

End;

procedure RemovePerson(Person: PPerson; WayOfRemoving: byte = 0);

Var

NextNode, CurrNode: PPerson;

begin

dec(AmountOfPersons);

// Проверка на наличие детей у персоны и второго родителя

if (Person^.Children <> nil) and (Person^.Spouse <> nil) then

Begin

CurrNode := Person^.Children;

case WayOfRemoving of

1:

While Assigned(CurrNode) do

Begin

if CurrNode^.Sibling <> nil then

NextNode := CurrNode^.Sibling;

Dispose(CurrNode);

dec(AmountOfPersons);

CurrNode := NextNode;

End;

2:

while CurrNode <> nil do

Begin

CurrNode^.Adopted := True;

CurrNode^.Mother := Person^.Mother;

CurrNode^.Father := Person^.Father;

CurrNode := CurrNode^.Sibling;

End;

3:

AddPerson(Person, nil, 'Spouse');

end;

End;

// Если не единственный ребенок в семье

if Person^.Mother <> nil then

if Person^.Mother^.Children <> Person then

begin

CurrNode := Person^.Mother^.Children;

While CurrNode^.Sibling <> Person do

CurrNode := CurrNode^.Sibling;

NextNode := Person^.Sibling;

CurrNode^.Sibling := NextNode;

end

Else

Begin

Person^.Mother^.Children := nil;

Person^.Father^.Children := nil;

End;

Dispose(Person);

end;

procedure GoingThroughTree(HeadOfTree: PPerson; var Visited: TStringList;

AOfProcess: Processes = nil);

var

i: Integer;

ind: string;

begin

ind := StringReplace(HeadOfTree^.Index, '@', '', [rfReplaceAll]);

// Проверяем, была ли данная персона уже посещена

if Visited.IndexOf(ind) = -1 then

Begin

// Посещаем данную персону

Visited.Add(ind);

for i := Low(AOfProcess) to High(AOfProcess) do

if Assigned(AOfProcess[i]) then

// Здесь можно выполнить какую-то операцию с посещенной персоной

AOfProcess[i](HeadOfTree);

// Рекурсивно обходим всех ее родителей

if Assigned(HeadOfTree^.Mother) then

GoingThroughTree(HeadOfTree^.Mother, Visited, AOfProcess);

// Рекурсивно обходим всех ее братьев и сестер

if Assigned(HeadOfTree^.Sibling) then

GoingThroughTree(HeadOfTree^.Sibling, Visited, AOfProcess);

// Рекурсивно обходим всех ее детей

if Assigned(HeadOfTree^.Children) then

GoingThroughTree(HeadOfTree^.Children, Visited, AOfProcess);

// Обходим супруга данной персоны, если он есть

if Assigned(HeadOfTree^.Spouse) then

GoingThroughTree(HeadOfTree^.Spouse, Visited, AOfProcess);

end;

end;

function GetFullName(Person: PPerson): string;

begin

if Person^.LName <> '' then

Begin

Result := Person^.LName + ' ';

End

else

Result := 'Неизвестно ';

if Person^.FName <> '' then

Begin

Result := Result + Person^.FName + ' ';

End

else

Result := Result + 'Неизвестно ';

if Assigned(Person^.Father) and (Person^.Father^.FName <> '') then

Begin

Result := Result + Person^.Father^.FName[1] + '.';

End;

end;

end.

Приложение В

(Обязательное)

Исходный код программы (модуль ForFiles)

unit uForFiles;

interface

uses umanipulwithperson, sysutils, classes;

procedure WriteIntoFile(StartPerson: PPerson; NeededFileP: String);

function ReadPersonsFromFile(const FileName: string): PPerson;

implementation

type

TFinPerson = record

Index: String[9];

Gender: TGender;

Mother, Father, Spouse, StepM, StepF: String[9];

Children: String[9];

Adopted: Boolean;

Sibling: String[9];

POfBirth: TLocation;

FName, LName, Note: String[100];

DOfBirth: TDateTime;

Photo: String[255];

Alive: Boolean;

ContactInfo: String[100];

POfDeath: TLocation;

DOfDeath: TDateTime;

end;

NeededF = File of TFinPerson;

TPersonArray = array of TFinPerson;

var

ResultFile: NeededF;

procedure PersonRelationToFin(const Person: PPerson);

var

NewP: TFinPerson;

begin

NewP.Index := Person^.Index;

NewP.Gender := Person^.Gender;

if assigned(Person^.Mother) then

NewP.Mother := Person^.Mother^.Index

Else

NewP.Mother := '@@';

if assigned(Person^.Father) then

NewP.Father := Person^.Father^.Index

Else

NewP.Father := '@@';

if assigned(Person^.Spouse) then

NewP.Spouse := Person^.Spouse^.Index

Else

NewP.Spouse := '@@';

if assigned(Person^.Children) then

NewP.Children := Person^.Children^.Index

Else

NewP.Children := '@@';

NewP.Adopted := Person^.Adopted;

if assigned(Person^.StepM) then

NewP.StepM := Person^.StepM^.Index

Else

NewP.StepM := '@@';

if assigned(Person^.StepF) then

NewP.StepF := Person^.StepF^.Index

Else

NewP.StepF := '@@';

if assigned(Person^.Sibling) then

NewP.Sibling := Person^.Sibling^.Index

Else

NewP.Sibling := '@@';

NewP.POfBirth := Person^.POfBirth;

NewP.FName := Person^.FName;

NewP.LName := Person^.LName;

NewP.DOfBirth := Person^.DOfBirth;

NewP.Photo := Person^.Photo;

NewP.Alive := Person^.Alive;

NewP.ContactInfo := Person^.ContactInfo;

NewP.POfDeath := Person^.POfDeath;

NewP.DOfDeath := Person^.DOfDeath;

Write(ResultFile, NewP);

end;

procedure PersonRelationBack(Person: TFinPerson; NewP: PPerson);

begin

NewP^.Index := Person.Index;

NewP^.Gender := TGender(Person.Gender);

NewP^.Mother := nil;

NewP^.Father := nil;

NewP^.Spouse := nil;

NewP^.Children := nil;

NewP^.Adopted := Person.Adopted;

NewP^.StepM := nil;

NewP^.StepF := nil;

NewP^.Sibling := nil;

NewP^.POfBirth := Person.POfBirth;

NewP^.FName := Person.FName;

NewP^.LName := Person.LName;

NewP^.DOfBirth := Person.DOfBirth;

NewP^.Photo := Person.Photo;

NewP^.Alive := Person.Alive;

if not Person.Alive then

begin

NewP^.ContactInfo := Person.ContactInfo;

NewP^.POfDeath := Person.POfDeath;

NewP^.DOfDeath := Person.DOfDeath;

end;

end;

procedure WriteIntoFile(StartPerson: PPerson; NeededFileP: string);

begin

AssignFile(ResultFile, NeededFileP);

Rewrite(ResultFile);

visit := TStringList.Create;

GoingThroughTree(StartPerson, visit, [@PersonRelationToFin]);

visit.Free;

CloseFile(ResultFile);

end;

procedure CreateLinkedListFromPersons(Persons: TPersonArray; var Root: PPerson);

var

i, g: Integer;

Person, Personbuf: PPerson;

indexesOfAll: array of string;

used: Boolean;

begin

if length(Persons) > 1 then

Begin

setlength(indexesOfAll, length(Persons));

for i := Low(Persons) to High(Persons) do

indexesOfAll[i] := Persons[i].Index;

for i := 0 to length(Persons) - 1 do

begin

if i = 0 then

Person := Root

else

new(Person);

PersonRelationBack(Persons[i], Person);

for g := i + 1 to High(Persons) do

begin

new(Personbuf);

PersonRelationBack(Persons[g], Personbuf);

used := false;

if samestr(indexesOfAll[g], (Persons[i]).Mother)

then

begin

AddPerson(Person, Personbuf, 'Mother');

used := true;

end;

if samestr(indexesOfAll[g], (Persons[i]).Father)

then

begin

AddPerson(Person, Personbuf, 'Father');

used := true;

end;

if samestr(indexesOfAll[g], (Persons[i]).StepM) then

begin

Person^.StepM := Personbuf;

AddPerson(Personbuf, Person, 'Mother');

used := true;

end;

if samestr(indexesOfAll[g], (Persons[i]).StepF) then

begin

Person^.StepF := Personbuf;

AddPerson(Personbuf, Person, 'Father');

used := true;

end;

if samestr(indexesOfAll[g], (Persons[i]).Spouse)

then

begin

AddPerson(Person, Personbuf, 'Spouse');

used := true;

end;

if samestr(indexesOfAll[g], (Persons[i]).Children)

then

begin

AddPerson(Person, Personbuf, 'Children');

used := true;

end;

if samestr(indexesOfAll[g], (Persons[i]).Sibling)

then

begin

Person^.Sibling := Personbuf;

used := true;

end;

if not used then

Dispose(Personbuf);

end;

end;

End;

end;

function ReadPersonsFromFile(const FileName: string): PPerson;

var

PersonFile: NeededF;

PersonCount, i: Integer;

Persons: TPersonArray;

begin

// Открытие файла для чтения

AssignFile(PersonFile, FileName);

Reset(PersonFile);

// Получение количества записей в файле

PersonCount := FileSize(PersonFile);

// Создание массива с нужным размером

setlength(Persons, PersonCount);

// Чтение записей из файла в массив

for i := 0 to PersonCount - 1 do

begin

Read(PersonFile, Persons[i]);

end;

CloseFile(PersonFile);

new(Result);

CreateLinkedListFromPersons(Persons, Result);

end;

end.

Приложение Г

(Обязательное)

Исходный код программы (модуль DeletingPerson)

unit uDeletePerson;

interface

uses

Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants,

System.Classes, Vcl.Graphics, uManipulWithPerson,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls, Vcl.ExtCtrls;

type

TFDeletePerson = class(TForm)

RGChooze: TRadioGroup;

BNo: TButton;

BYes: TButton;

LConfrm: TLabel;

procedure DeletePersonShow(Person: PPerson);

procedure BYesClick(Sender: TObject);

procedure BNoClick(Sender: TObject);

procedure RGChoozeClick(Sender: TObject);

end;

var

FDeletePerson: TFDeletePerson;

implementation

var

PersonForDeleting: PPerson;

Lcaption: string = 'Вы уверены, что хотите удалить ';

{$R \*.dfm}

procedure TFDeletePerson.BNoClick(Sender: TObject);

begin

ModalResult := mrOk;

end;

procedure TFDeletePerson.BYesClick(Sender: TObject);

begin

BYes.Visible := False;

BNo.Visible := False;

if (PersonForDeleting^.Children <> nil) and (PersonForDeleting^.Spouse = nil)

then

Begin

LConfrm.Visible := False;

RGChooze.Visible := True;

End

else

Begin

RemovePerson(PersonForDeleting);

LConfrm.caption := 'Персона удалена';

End;

end;

procedure TFDeletePerson.DeletePersonShow(Person: PPerson);

begin

PersonForDeleting := Person;

LConfrm.caption := Lcaption + GetFullName(PersonForDeleting);

BYes.Visible := True;

BNo.Visible := True;

FDeletePerson.ShowModal;

end;

procedure TFDeletePerson.RGChoozeClick(Sender: TObject);

begin

RemovePerson(PersonForDeleting, RGChooze.ItemIndex);

ModalResult := mrOk;

end;

end.