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)

ОТЧЕТ

по учебной практике

Выполнил

студент: гр. 251003 Панкратьев Е.С.

Проверил: Фадеева Е.П.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Постановка задачи 3](#_Toc136279644)

[2 Методика решения 4](#_Toc136279645)

[2.1 Описание средств разработки 4](#_Toc136279646)

[2.1.1 Линейный двунаправленный список 4](#_Toc136279647)

[2.1.2 Файлы 5](#_Toc136279648)

[2.2 Спецификация функциональных требований 7](#_Toc136279649)

[3 Описание алгоритмов решения задачи 8](#_Toc136279650)

[4 Структура данных 10](#_Toc136279651)

[4.1 Структура типов программы 10](#_Toc136279652)

[4.2 Структура типов программы 11](#_Toc136279653)

[4.3 Структура данных алгоритма AddNode 11](#_Toc136279654)

[4.4 Структура данных алгоритма Clear 11](#_Toc136279655)

[4.5 Структура данных алгоритма FindNodeByName 12](#_Toc136279656)

[4.6 Структура данных алгоритма SortByParameter 12](#_Toc136279657)

[4.7 Структура данных алгоритма SaveDataToFile 13](#_Toc136279658)

[4.8 Структура данных алгоритма LoadDataFromFile 13](#_Toc136279659)

[4.9 Структура данных алгоритма CheckCompatibility 13](#_Toc136279660)

[5 Схема алгоритма решения задачи по ГОСТ 19.701-90 14](#_Toc136279661)

[5.1 Схема алгоритма SaveDataToFile 14](#_Toc136279662)

[5.2 Схема алгоритма Clear 15](#_Toc136279663)

[5.3 Схема алгоритма FindNodeByName 16](#_Toc136279664)

[5.4 Схема алгоритма SortByParameter 17](#_Toc136279665)

[5.5 Схема алгоритма SaveDataToFile 19](#_Toc136279666)

[5.6 Схема алгоритма LoadDataFromFile 21](#_Toc136279667)

[5.7 Схема алгоритма CheckCompatib 23](#_Toc136279668)

[6 Графический интерфейс 25](#_Toc136279669)

[6.1 Описание графических компонентов формы frmMain 25](#_Toc136279670)

[6.2 Описание графических компонентов формы frmInfo 26](#_Toc136279671)

[7 Тестирование и проверка работоспособности программного средства 28](#_Toc136279672)

[7.1 Запуск программы 28](#_Toc136279673)

[7.1.1 Тест 1 28](#_Toc136279674)

[7.2 Добавление кандидата 28](#_Toc136279675)

[7.2.1 Тест 1 28](#_Toc136279676)

[7.2.2 Тест 2 29](#_Toc136279677)

[7.3 Удаление кандидата 29](#_Toc136279678)

[7.3.1 Тест 1 29](#_Toc136279679)

[7.4 Сортировка 30](#_Toc136279680)

[7.4.1 Тест 1 30](#_Toc136279681)

[7.4.2 Тест 2 31](#_Toc136279682)

[Приложение А 32](#_Toc136279683)

[Приложение Б 47](#_Toc136279684)

[Приложение В 51](#_Toc136279685)

# Постановка задачи

Разработать программу («электронную сваху») для службы знакомств. Имеется 2 списка: список женихов и список невест. В каждом списке кандидат (жених или невеста) характеризуются записью, содержащей следующие поля: порядковый номер кандидата, данные о кандидате (имя, возраст, рост, вес, привычки, хобби), требования к партнеру (в виде диапазона Min-Мах для возраста, роста, веса). Требуется:

* выявить все возможные пары с учетом требований кандидатов;
* при согласии сторон пара считается сформированной и кандидаты в списке помечаются как удаленные.

# Методика решения

## Описание средств разработки

В данном программном средстве будут использоваться две основные структуры данных:

* линейный двунаправленный список;
* файлы.

### Линейный двунаправленный список

Двунаправленный список - это динамическая структура данных, состоящая из узлов одного типа, связанных между собой в двух направлениях: вперед и назад. Каждый узел содержит данные и указатели на предыдущий и следующий узлы в списке.

Узел двунаправленного списка содержит данные и два указателя: на предыдущий и следующий узлы. Первый узел списка - головной узел, последний - хвостовой узел. Указатель на предыдущий узел головного узла и указатель на следующий узел хвостового узла равны Nil.

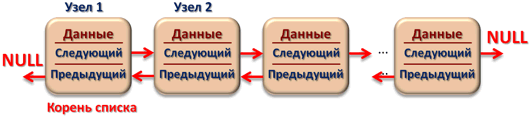


Рисунок 2.1 – Иллюстрация структуры линейного однонаправленного связного списка

Двунаправленный список создается путем инициализации головного и хвостового узлов. Указатели на предыдущий и следующий узлы в узлах устанавливаются в Nil.

Добавление нового узла в список включает создание нового узла, установку указателей на предыдущий и следующий узлы, а также обновление указателей в соседних узлах.

Удаление узла из списка требует обновления указателей в соседних узлах, чтобы они обходили удаленный узел. После этого освобождается память, занимаемая удаленным узлом.

Поиск узла по значению выполняется путем последовательного обхода узлов списка и сравнения значений. Если найдено совпадение, возвращается указатель на найденный узел. Если ни один узел не соответствует искомому значению, возвращается Nil.

Обход списка может осуществляться в прямом и обратном направлениях. При прямом обходе начинают с головного узла и последовательно переходят к следующим узлам, выполняя необходимые операции. При обратном обходе начинают с хвостового узла и двигаются к предыдущим узлам.Двунаправленный список - это удобная структура данных для хранения и организации элементов в двух направлениях. Он позволяет эффективно выполнять операции добавления, удаления, поиска и обхода элементов. Использование двунаправленного списка помогает эффективно управлять данными и решать различные задачи программирования.

Программа использует двунаправленный список для хранения списка женихов и невест. Выбор двунаправленного списка обусловлен несколькими преимуществами.

Одно из основных преимуществ двунаправленного списка - эффективное удаление элементов. При удалении элемента из списка, необходимо лишь обновить указатели в соседних узлах, что позволяет осуществлять удаление за постоянное время O(1), независимо от размера списка.

Кроме того, двунаправленный список обеспечивает простой доступ к предыдущему узлу. Это позволяет удобно выполнять операции и обходы списка, так как предыдущий узел доступен непосредственно через указатель в каждом узле, в отличие от однонаправленного списка, где доступ к предыдущему узлу требует выполнения обратного поиска.

Таким образом, использование двунаправленного списка в данной программе обеспечивает эффективное удаление элементов и удобный доступ к предыдущему узлу, что может быть важным для реализации требуемых операций и обходов списка.

### Файлы

Файловый тип в программировании представляет собой структуру данных, которая содержит произвольную последовательность элементов. Его длина не определяется заранее, а конкретизируется в процессе выполнения программы. Файловый тип используется для организации хранения и обработки данных на внешних носителях, таких как жесткий диск или флеш-память.

В языке программирования Delphi существуют три типа файлов:

* типизированные файлы: они связываются с файловыми переменными, объявленными как file of <Тип>. Файл такого типа считается состоящим из элементов, каждый из которых имеет определенный тип данных;
* текстовые файлы: они представляют собой последовательность символов и связываются с файловыми переменными типа TextFile. Текстовые файлы имеют особенность разбиваться на строки, где каждая строка может иметь различную длину и заканчивается специальным символом, обозначающим конец строки;
* нетипизированные файлы: они могут быть связаны только с файловыми переменными, объявленными как file. Файл такого типа считается состоящим из элементов, размер которых определяется при открытии файла. В нетипизированных файлах данные могут быть организованы по свободному формату.

Для работы с файлами в Delphi существуют стандартные процедуры и функции ввода-вывода, которые обеспечивают доступ к отдельным элементам файлов. Эти функции позволяют осуществлять чтение, запись, поиск и другие операции над файлами. Использование соответствующих процедур и функций облегчает взаимодействие с файловой системой и обработку данных в программах.

Таблица 1 – Основные процедуры и функции для работы с файлами

| Имя | Вид | Описание | Тип результата |
| --- | --- | --- | --- |
| AssignFile (F, Path) | Процедура | Связывает файловую переменную F с файлом по пути Path | – |
| Append (F) | Процедура | Открывает существующий файл F и ставит указатель в конец файла | – |
| CloseFile (F) | Процедура | Закрывает файл F и сохраняет изменения | ­– |
| Eof (F) | Функция | Проверяет, стоит ли указатель в конце файла F | Boolean |
| Eoln (F) | Функция | Проверяет, стоит ли указатель в конце строки файла F | Boolean |
| FilePos (F) | Функция | Возвращает текущую позицию указателя в файле F | LongInt |
| Read (F, V1,…Vn) | Процедура | Считывает из файла F значения и записывает их в переменные V1, …, Vn | – |
| Rename (F, Name) | Процедура | Переименовывает файл F в Name | – |
| Seek (F, N) | Процедура | Устанавливает указатель на позицию N в файле F | – |
| Write (F, V1, …, Vn) | Процедура | Записывает в файл значения V1, …, Vn | – |

В электронной свахе будут сохраняться данные о списках женихов и невест. В каждой записи будут содержаться следующие поля на русском языке:

* номер серии: целочисленное значение, которое идентифицирует каждую запись в списке;
* имя: краткая строка, содержащая имя жениха или невесты;
* возраст: целочисленное значение, указывающее на возраст жениха или невесты;
* рост: целочисленное значение, представляющее рост жениха или невесты в сантиметрах;
* вес: целочисленное значение, указывающее на вес жениха или невесты в килограммах;
* привычки: краткая строка, описывающая особые привычки жениха или невесты;
* хобби: краткая строка, указывающая на основное хобби жениха или невесты;
* пол: логическое значение, показывающее пол жениха или невесты;
* предпочтения: структура, содержащая минимальный и максимальный возраст, рост и вес, указывающие предпочтения жениха или невесты по возрасту, росту и весу.

Таким образом, каждая запись в электронной свахе будет содержать информацию о женихе или невесте, включая их личные данные, особенности и предпочтения.

## Спецификация функциональных требований

После анализа условия задачи был составлен список функциональных требований к программному средству, представленный в таблице 2.

Таблица 2 – Функциональные требования к программному средству

|  |  |
| --- | --- |
| Идентификатор | Требование |
| ФТ-1 | Создать возможность добавления жениха или невесты в список |
| ФТ-2 | Создать возможность удаления жениха или невесты из списка |
| ФТ-3 | Создать возможность редактирования информации о женихе или невесте в списке |
| ФТ-4 | Создать возможность сохранения списка женихов и невест в типизированный файл |
| ФТ-5 | Создать возможность сохранения списка женихов и невест в типизированный файл |
| ФТ-6 | Создать возможность чтения списка женихов и невест из типизированного файла |
| ФТ-7 | Создать возможность сортировки списка по различным параметрам (например, возраст, рост, вес) |

# Описание алгоритмов решения задачи

Таблица 3 – Описание алгоритмов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| №  п.п. | Наименование алгоритма | Назначение алгоритма | Формальные параметры | Предлагаемый тип реализации |
| 1 | Основной алгоритм | Вызывает следующие подпрограммы:  AddNode, Clear,  FindNodeByName,  SortByParameter,  DeleteFootballer,  SaveDataToFile,  LoadDataToFile,  CheckCompatibility |  |  |
| 2 | AddNode  (  Self,  AData  ) | Добавляет в конец списка Self новое звено с данными AData | Self – получает от фактического параметра адрес с защитой;  AData – получает от фактического параметра адрес с защитой | Процедура |
| 3 | Clear  (  Self  ) | Удаляет все узлы из списка Self | Self – получает от фактического параметра адрес с защитой | Процедура |
| 4 | FindNodeBy-  Name  (  Self,  AName,  Res  ) | Находит узел в списке Self по названию AName | Self – получает от фактического параметра адрес с защитой;  AName – получает от фактического параметра адрес с защитой;  Res – получает от фактического параметра адрес, возвращаемый параметр | Функция (Res – возвращаемый функцией параметр) |
| 5 | SortByPara-meter | Сортирует список Self с | Self – получает от фактического | Процедура |

Продолжение таблицы 3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | (  Self,  Compare  ) | использованием компаратора Compare | параметра адрес с защитой;  Compare – получает от фактического параметра адрес с защитой |  |
| 6 | SaveData-ToFile  (  List,  FileName  ) | Сохраняет данные списка List в типизированный файл, используя заданный путь FileName | List – получает от фактического параметра адрес с защитой;  FileName – получает от фактического параметра адрес с защитой | Процедура |
| 7 | LoadData-  FromFile  (  List,  FileName  ) | Открывает типизированный файл по заданному пути FileName и загружает данные в список List | List – получает от фактического параметра адрес с защитой;  FileName – получает от фактического параметра адрес с защитой | Процедура |
| 8 | CheckCompa-tibility  (  First,  Second,  Res  ) | Проверяет совместимость между First и Second. Записывает результат в переменную Res | First – получает от фактического параметра адрес с защитой;  Second – получает от фактического параметра адрес с защитой;  Res – получает от фактического параметра адрес, возвращаемый параметр | Функция (Res – возвращаемый функцией параметр) |

# Структура данных

## Структура типов программы

Таблица 4 – Структура типов программы

|  |  |  |
| --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение |
| TPreference | Record  minAge: Integer;  maxAge: Integer;  minHeight: Integer;  maxHeight: Integer;  minWeight: Integer;  maxWeight: Integer;  end; | Хранение предпочтений (параметров) в отношении возраста, роста и веса |
| TNodeData | Record  SerialNum: Integer;  Name: ShortString;  Age: Integer;  Height: Integer;  Weight: Integer;  Habits: ShortString;  Hobby: ShortString;  isMale: Boolean;  Preference: TPreference;  end; | Данные узла, которые хранят информацию об человеке |
| PList | ^TNode | Предоставляет структуру для хранения данных об человеке в контексте двунаправленного списка. Каждый узел содержит ссылки на предыдущий и следующий узлы. Кроме того, узел хранит информацию о человеке, которая может быть связана с ним |
| TNode | Record  Data: TNodeData;  Prev: PNode;  Next: PNode;  end; |
| TCompare-Function | Function(const AFirst, ASecond: TNodeData): Boolean | Используется для определения сравнения двух элементов типа TNodeData |

Продолжение таблицы 4

|  |  |  |
| --- | --- | --- |
| TDoubly-LinkedList | Record  FHead: PNode;  FTail: PNode;  end; | Представляет структуру двусвязанного списка. Хранит ссылку на первый и последний элемент |

## Структура типов программы

Таблица 5 – Структура данных программы

|  |  |  |
| --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение |
| MaleList | TDoublyLinkedList | Указатель на линейный двунаправленный список женихов |
| FemaleList | TDoublyLinkedList | Указатель на линейный двунаправленный список невест |

## Структура данных алгоритма AddNode

Таблица 6 – Структура данных алгоритма AddNode(Self, AData)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Self | TDoublyLinked-List | Указатель на линейный двунаправленный список | Формальный |
| AData | TNodeData | Информация об человеке | Формальный |
| NewNode | PNode | Адрес нового узла | Локальный |

## Структура данных алгоритма Clear

Таблица 7 – Структура данных алгоритма Clear(Self)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Self | TDoublyLinked-List | Указатель на линейный двунаправленный список | Формальный |
| CurrentNode | PNode | Адрес текущего узла | Локальный |
| NextNode | PNode | Адрес следующего узла | Локальный |

## Структура данных алгоритма FindNodeByName

Таблица 8 – Структура данных алгоритма FindNodeByName(Self, AName, Res)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Self | TDoublyLinked-List | Указатель на линейный двунаправленный список | Формальный |
| AName | String | Имя, по которому необходимо найти узел | Формальный |
| Res | PNode | Найденный узел | Формальный |
| CurrentNode | PNode | Адрес текущего узла | Локальный |

## Структура данных алгоритма SortByParameter

Таблица 9 – Структура данных алгоритма SortByParameter(Self, Compare)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Self | TDoublyLinked-List | Указатель на линейный двунаправленный список | Формальный |
| Compare | TCompareFunction | Используется для сравнения узлов списка | Формальный |
| Current | PNode | Адрес текущего узла в списке | Локальный |
| Running | PNode | Адрес прогоняемого узла в списке | Локальный |
| Support | PNode | Адрес опорного узла в списке | Локальный |
| TempData | TNodeData | Временная переменная для обмена данных | Локальный |
| Swapped | Boolean | Индикатор обмена данных при каждом проходе по списку | Локальный |

## Структура данных алгоритма SaveDataToFile

Таблица 10 ­– Структура данных алгоритма SaveDataToFile(List, FileName)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| List | TDoublyLinked-List | Указатель на линейный двунаправленный список | Формальный |
| FileName | String | Путь к файлу для сохранения | Формальный |
| DataFile | file of TNodeData | Файл для хранения данных | Локальный |
| NodeData | TNodeData | Информация об текущем узле | Локальный |

## Структура данных алгоритма LoadDataFromFile

Таблица 11 – Структура данных алгоритма LoadDataFromFile(List, FileName)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| List | TDoublyLinked-List | Указатель на линейный двунаправленный список | Формальный |
| FileName | String | Путь к файлу для чтения | Формальный |
| DataFile | file of TNodeData | Файл для чтения данных | Локальный |
| NodeData | TNodeData | Информация об текущем узле | Локальный |

## Структура данных алгоритма CheckCompatibility

Таблица 12 – Структура данных алгоритма CheckCompatib(First, Second, Res)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| First | TNodeData | Первый узел для проверки | Формальный |
| Second | TNodeData | Второй узел для проверки | Формальный |
| Res | Boolean | Результат проверки совместимости | Формальный |

# Схема алгоритма решения задачи по ГОСТ 19.701-90

## Схема алгоритма SaveDataToFile



Рисунок 5.1 – Схема алгоритма SaveDataToFile

## Схема алгоритма Clear



Рисунок 5.2 – Схема алгоритма Clear

## Схема алгоритма FindNodeByName



Рисунок 5.3 – Схема алгоритма FindNodeByName

## Схема алгоритма SortByParameter



Рисунок 5.4 – Схема алгоритма SortByParameter (часть 1)



Рисунок 5.5 – Схема алгоритма SortByParameter (часть 2)

## Схема алгоритма SaveDataToFile



Рисунок 5.6 – Схема алгоритма SaveDataToFile (часть 1)



Рисунок 5.7 – Схема алгоритма SaveDataToFile (часть 2)

## Схема алгоритма LoadDataFromFile



Рисунок 5.8 – Схема алгоритма LoadDataFromFile (часть 1)



Рисунок 5.9 – Схема алгоритма LoadDataFromFile (часть 2)

## Схема алгоритма CheckCompatib



Рисунок 5.10 – Схема алгоритма CheckCompatib (часть 1)



Рисунок 5.11 – Схема алгоритма CheckCompatib (часть 2)

# Графический интерфейс

## Описание графических компонентов формы frmMain

Форма frmMain – это основная форма, которая даёт пользователю доступ к основным функциям программного средста, таким как просмотр списков невест и женихов, добавление, редактирование или удаление кандидатов, сортировка списков по различным параметрам. Данная форма имеет вид, представленный на рисунке 6.1.

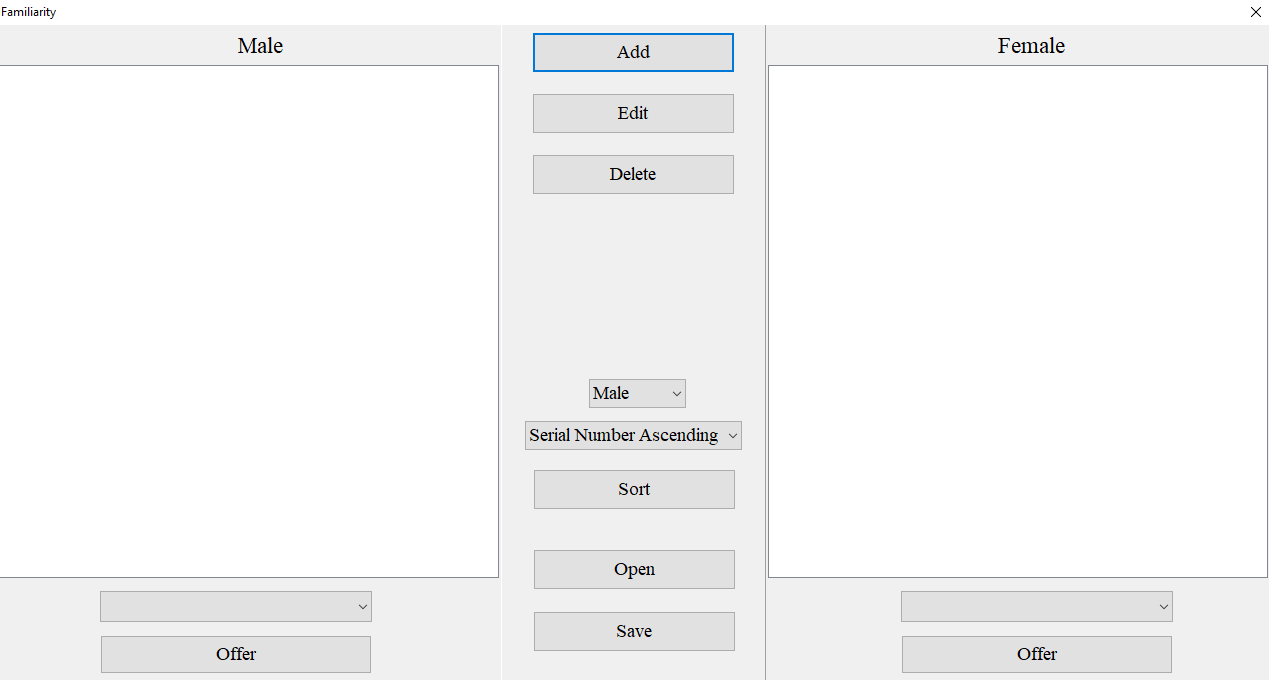


Рисунок 6.1 – Вид формы frmMain

Составляющие формы frmMain:

* список «lvMale», на которой отображается женихи;
* список «lvFemale», на которой отображается невестки;
* кнопка «btnAdd», отвечающая за добавление нового кандидата;
* кнопка «btnDel», отвечающая за удаление выбранного кандидата;
* кнопка «btnEdit», отвечающая за редактирование информации об кандидате;
* компонент «cbSortGender», отвечающий за определения того, какой список сортировать;
* компонент «cbSort», отвечающий за определния того, по какому принципу сортировать список;
* кнопка «btnSort», отвечающая за сортировку;
* кнопка «btnOpen», отвечающая за открытие файла;
* кнопка «btnSave», отвечающая за сохранения файла;
* компоненты «cbMail», «cbFemail», отвечающие за выбора кандидата;
* кнопки «btnMaleOffer», «btnFemaleOffer», отвечающие за предложение об браке.

## Описание графических компонентов формы frmInfo

Форма frmInfo позволяет пользователю добавлять новых кандидатов или редактировать уже существующих. Данная форма имеет вид, представленный на рисунках 6.2.

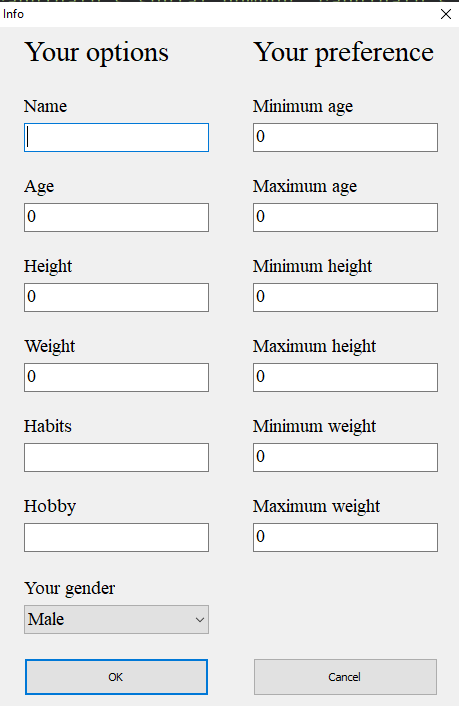


Рисунок 6.2 – Вид формы frmInfo

Составляющие формы frmInfo:

* кнопка «btnOK», позволяющая добавить или сохранить информацию об кандидате;
* кнопка «btnCancel», позволяющая выйти из формы;
* компонент «cdGender», позволяющий определить гендер кандидата;
* поле для ввода «edtName» предназначено для ввода имени кандидата;
* поле для ввода «edtAge» предназначено для ввода возраста кандидата;
* поле для ввода «edtHeight» предназначено для ввода роста кандидата;
* поле для ввода «edtWeight» предназначено для ввода веса кандидата;
* поле для ввода «edtHabits» предназначено для ввода привычек или характеристик кандидата;
* поле для ввода «edtHobby» предназначено для ввода хобби кандидата;
* поле для ввода «edtMinAge» предназначено для указания предпочтения минимального значения возраста;
* поле для ввода «edtMaxAge» предназначено для указания предпочтения максимального значения возраста;
* поле для ввода «edtMinHeight» предназначено для указания предпочтения минимального значения роста;
* поле для ввода «edtMaxHeight» предназначено для указания предпочтения максимального значения роста;
* поле для ввода «edtMinWeight» предназначено для указания предпочтения минимального значения веса;
* поле для ввода «edtMaxWeight» предназначено для указания предпочтения максимального значения веса.

# Тестирование и проверка работоспособности программного средства

## Запуск программы

### Тест 1

Таблица 13 – Тест 1

|  |  |
| --- | --- |
| Тестовая ситуация: | Проверка корректности поведения программы при запуске |
| Исходный набор данных: | Запуск программы |
| Ожидаемый результат: | Открытие формы frmMain, отображение списков |
| Полученный результат: |  |

## Добавление кандидата

### Тест 1

Таблица 14 – Тест 1

|  |  |
| --- | --- |
| Тестовая ситуация: | Проверка корректности работы программы при добавлении кандидата |
| Исходный набор данных: | Нажатие на кнопку btnAdd |
| Ожидаемый результат: | Добавление кандидата |

Продолжение таблицы 14

|  |  |
| --- | --- |
| Полученный результат: |  |

### Тест 2

Таблица 15 – Тест 2

|  |  |
| --- | --- |
| Тестовая ситуация: | Проверка корректности работы программы при добавлении кандидата с некорректными данными |
| Исходный набор данных: | Нажатие на кнопку btnAdd |
| Ожидаемый результат: | Просьба ввести корректные данные |
| Полученный результат: |  |

## Удаление кандидата

### Тест 1

Таблица 16 – Тест 1

|  |  |
| --- | --- |
| Тестовая ситуация: | Проверка корректности работы программы при нажатии на кнопку добавления нового футболиста |
| Исходный набор данных: | Нажатие на кнопку btnDelete |
| Ожидаемый результат: | Удаление кандидата |

Продолжение таблицы 16

|  |  |
| --- | --- |
| Полученный результат: |  |

## Сортировка

### Тест 1

Таблица 17 – Тест 1

|  |  |
| --- | --- |
| Тестовая ситуация: | Проверка корректности поведения программы при сортировке имен по возрастанию |
| Исходный набор данных: | Нажатие на кнопку btnSort |
| Ожидаемый результат: | Отсортированный список |
| Полученный результат: |  |

### Тест 2

Таблица 18 – Тест 2

|  |  |
| --- | --- |
| Тестовая ситуация: | Проверка корректности поведения программы при сортировке имен по убыванию |
| Исходный набор данных: | Нажатие на кнопку btnSort |
| Ожидаемый результат: | Отсортированный список |
| Полученный результат: |  |

Приложение А

(обязательное)

**Текст программного модуля frmMain**

unit frmMain;

{

The program will have two lists: a list of grooms and a

list of brides.

Each candidate (groom or bride) in the list will be

characterized by the

following fields: candidate's serial number, candidate's

information (name,

age, height, weight, habits, hobbies), and partner

preferences (in the form

of a Min-Max range for age, height, and weight).

}

interface

uses

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

System.Variants, System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.ComCtrls,

Vcl.Grids, Vcl.ExtCtrls,

Vcl.StdCtrls, uTypes, frmInfo, Vcl.Imaging.jpeg,

System.UITypes;

type

TMain = class(TForm)

Panel: TPanel;

btnAdd: TButton;

btnDelete: TButton;

lvMale: TListView;

lvFemale: TListView;

btnEdit: TButton;

lbMale: TLabel;

lbFemale: TLabel;

cbMale: TComboBox;

cbFemale: TComboBox;

btnMaleOffer: TButton;

btnFemaleOffer: TButton;

Image: TImage;

SaveDialog: TSaveDialog;

OpenDialog: TOpenDialog;

btnOpen: TButton;

btnSave: TButton;

cbSort: TComboBox;

cbSortGender: TComboBox;

procedure FormCreate(Sender: TObject);

procedure btnAddClick(Sender: TObject);

procedure btnDeleteClick(Sender: TObject);

procedure btnEditClick(Sender: TObject);

procedure lvMaleChange(Sender: TObject; Item: TListItem;

Change: TItemChange);

procedure lvFemaleChange(Sender: TObject; Item: TListItem;

Change: TItemChange);

procedure btnMaleOfferClick(Sender: TObject);

procedure btnFemaleOfferClick(Sender: TObject);

procedure SaveDataToFile(Sender: TObject);

procedure LoadDataFromFile(Sender: TObject);

procedure ImageClick(Sender: TObject);

private const

// The output message displayed when confirming a

// mutual liking

Output = ', you liked the person. Click yes if you like

it too.';

// The file extension for the Oi files

OIExt = '.oi';

// The file filter description for the Open and Save

// dialogs

FMOI = 'Oi Files (\*' + OIExt + ')|\*' + OIExt;

// An array of sorting functions used for sorting the

// data.

// Each function corresponds to a specific sorting

// criteria

SortingFunctions: array[0..13] of TCompareFunction = (

CompareBySerialNumAsc,

CompareBySerialNumDesc,

CompareByNameAsc,

CompareByNameDesc,

CompareByAgeAsc,

CompareByAgeDesc,

CompareByHeightAsc,

CompareByHeightDesc,

CompareByWeightAsc,

CompareByWeightDesc,

CompareByHabitsAsc,

CompareByHabitsDesc,

CompareByHobbyAsc,

CompareByHobbyDesc

);

private

{ Private declarations }

MaleList, FemaleList: TDoublyLinkedList;

procedure UpdateMaleList;

procedure UpdateFemaleList;

class function CheckCompatibility(const First, Second:

TNodeData): Boolean; static;

class function CheckCompatibilityHelper(const First,

Second: TNodeData): Boolean; static;

class function GetNodeDataDescription(const NodeData:

TNodeData): string; static;

public

{ Public declarations }

destructor Destroy; override;

end;

var

Main: TMain;

implementation

{$R \*.dfm}

// Updates the male list

procedure TMain.UpdateMaleList;

var

CurrentNode: PNode;

NodeData: TNodeData;

begin

lvMale.Clear;

CurrentNode := MaleList.Head;

// Clear the male list view before updating it.

lvMale.Clear;

// Traverse through each node in the MaleList.

while CurrentNode <> nil do

begin

NodeData := CurrentNode.Data;

// Add the name of the node data as an item in the

// male list view.

lvMale.AddItem(NodeData.Name, nil);

CurrentNode := CurrentNode.Next;

end;

end;

// Updates the female list

procedure TMain.UpdateFemaleList;

var

CurrentNode: PNode;

NodeData: TNodeData;

begin

lvFemale.Clear;

CurrentNode := FemaleList.Head;

// Clear the female list view before updating it.

lvFemale.Clear;

// Traverse through each node in the FemaleList.

while CurrentNode <> nil do

begin

NodeData := CurrentNode.Data;

// Add the name of the node data as an item in the

// female list view.

lvFemale.AddItem(NodeData.Name, nil);

CurrentNode := CurrentNode.Next;

end;

end;

// Event for deleting a node

procedure TMain.btnDeleteClick(Sender: TObject);

begin

// Check if a male item is selected in the list view.

if lvMale.Selected <> nil then

begin

// Remove the selected male node from the MaleList

// and delete the corresponding item from the list

// view.

MaleList.RemoveNode(MaleList.FindNodeByName(lvMale.Selected

.Caption));

lvMale.Items.Delete(lvMale.Selected.Index);

end;

// Check if a female item is selected in the list view.

if lvFemale.Selected <> nil then

begin

// Remove the selected female node from the

// FemaleList and delete the corresponding item from

// the list view.

FemaleList.RemoveNode(FemaleList.FindNodeByName(lvFemale.Selected.Caption));

lvFemale.Items.Delete(lvFemale.Selected.Index);

end;

end;

// Event for node change

procedure TMain.btnEditClick(Sender: TObject);

var

Node: PNode;

begin

// Check if a male item is selected in the list view.

if lvMale.Selected <> nil then

begin

// Find the selected male node in the MaleList.

Node := MaleList.FindNodeByName(lvMale.Selected.

Caption);

if Info.TryGetNodeData(Node^.Data) then

// Update the caption of the selected item in the

// male list

// view with the updated name from the node data.

lvMale.Selected.Caption := Node^.Data.Name;

end;

// Check if a female item is selected in the list view.

if lvFemale.Selected <> nil then

begin

// Find the selected female node in the FemaleList.

Node := FemaleList.FindNodeByName(lvFemale.Selected.

Caption);

if Info.TryGetNodeData(Node^.Data) then

// Update the caption of the selected item in the

// female list

// view with the updated name from the node data.

lvFemale.Selected.Caption := Node^.Data.Name;

end;

end;

// Destroy the main form

destructor TMain.Destroy;

begin

// Destroy the MaleList and FemaleList instances

MaleList.Destroy;

FemaleList.Destroy;

end;

// Event for adding node

procedure TMain.btnAddClick(Sender: TObject);

var

NodeData: TNodeData;

begin

// Initialize NodeData with the default values of

// TNodeData.

NodeData := Default(TNodeData);

// Try to get the node data from the Info component.

if Info.TryGetNodeData(NodeData) then

begin

// Check the selected gender in the Info component.

if Boolean(Info.cbGender.ItemIndex) then

begin

// Add the NodeData as a new node to the

// FemaleList.

FemaleList.AddNode(NodeData);

// Add a new item with the name of NodeData to the

// lvFemale list view.

lvFemale.AddItem(NodeData.Name, nil);

end

else

begin

// Add the NodeData as a new node to the MaleList.

MaleList.AddNode(NodeData);

// Add a new item with the name of NodeData to the

// lvMale list view.

lvMale.AddItem(NodeData.Name, nil);

end;

end;

end;

// Creating the main form

procedure TMain.FormCreate(Sender: TObject);

begin

// Create instances of TDoublyLinkedList for MaleList

// and FemaleList.

MaleList := TDoublyLinkedList.Create;

FemaleList := TDoublyLinkedList.Create;

// Set the default file extensions for save and open

// dialogs.

SaveDialog.DefaultExt := OIExt;

OpenDialog.DefaultExt := OIExt;

// Set the filters for save and open dialogs to display

// only OI files.

SaveDialog.Filter := FMOI;

OpenDialog.Filter := FMOI;

// Position the form at the center of the screen.

Left := (Screen.Width - Width) shr 1;

Top := (Screen.Height - Height) shr 1;

end;

class function TMain.CheckCompatibility(const First, Second: TNodeData): Boolean;

begin

// Check the compatibility between First and Second by

// calling the CheckCompatibilityHelper

// for both combinations: First with Second and Second

// with First.

Result := CheckCompatibilityHelper(First, Second) and

CheckCompatibilityHelper(Second, First);

end;

class function TMain.CheckCompatibilityHelper(const

First, Second: TNodeData): Boolean;

begin

// Check the compatibility between First and Second

// based on their

// attributes and preferences. Return True if the

// attributes of

// First (age, height, weight) fall within the range of

// preferences

// specified

Result := (First.Age >= Second.Preference.MinAge) and

(First.Age <= Second.Preference.MaxAge) and

(First.Height >= Second.Preference.MinHeight)

and

(First.Height <= Second.Preference.MaxHeight)

and

(First.Weight >= Second.Preference.MinWeight)

and

(First.Weight <=

Second.Preference.MaxWeight);

end;

procedure TMain.btnFemaleOfferClick(Sender: TObject);

var

Answer, I: integer;

MainNode, CurrNode: PNode;

begin

if cbFemale.ItemIndex > -1 then

begin

// Get the main female node based on the selected

// item in the lvFemale list view.

MainNode := FemaleList.FindNodeByName(lvFemale.

Selected.Caption);

// Get the corresponding male node based on the

// selected item in the cbFemale combo box.

CurrNode := MaleList.FindNodeByName(cbFemale.

Items.Strings[cbFemale.ItemIndex]);

// Display a message dialog to offer the male node to

// the female, showing relevant information.

Answer := MessageDlg(CurrNode.Data.Name + Output +

sLineBreak +

GetNodeDataDescription(MainNode.Data),

mtInformation, [mbYes, mbNo], 0);

case Answer of

mrYes:

begin

// If the female accepts the offer (clicked "Yes"

// button), proceed with the following actions:

// Remove the female node from the FemaleList.

FemaleList.RemoveNode(MainNode);

// Delete the corresponding item from the

// lvFemale list view.

lvFemale.Items.Delete(lvFemale.Selected.Index);

// Iterate through the lvMale list view to find

// the item matching the male node.

// Delete that item from the lvMale list view.

for I := 0 to lvMale.Items.Count - 1 do

if lvMale.Items[I].Caption = CurrNode.Data.Name

then

begin

lvMale.Items.Delete(I);

Break;

end;

// Remove the male node from the MaleList.

MaleList.RemoveNode(CurrNode);

// Update the male and female list views to

// reflect the changes.

Self.UpdateMaleList;

Self.UpdateFemaleList;

end;

end;

end;

end;

procedure TMain.lvFemaleChange(Sender: TObject; Item: TListItem; Change: TItemChange);

var

I: Integer;

MainNode, CurrNode: TNodeData;

begin

cbFemale.Items.Clear;

// Check if there are items in lvFemale and lvMale

// lists, and if a female item is selected.

if (lvFemale.Items.Count > 0) and (lvMale.Items.Count >

0) and (lvFemale.Selected <> nil) then

begin

// Retrieve the data of the selected female node.

MainNode := FemaleList.FindNodeByName(lvFemale.

Selected.Caption).Data;

// Iterate through the lvMale items and check

// compatibility with the selected female node.

for I := 0 to lvMale.Items.Count - 1 do

begin

// Retrieve the data of the current male node.

CurrNode := MaleList.FindNodeByName(lvMale.

Items[I].Caption).Data;

// Check compatibility between the selected female

// node and the current male node.

// If compatible, add the current male node's name

// to the cbFemale combo box.

if CheckCompatibility(MainNode, CurrNode) then

cbFemale.Items.Add(CurrNode.Name);

end;

end;

// Trigger the lvMaleChange event to update the male

// compatibility information.

if Sender <> nil then

lvMaleChange(nil, Item, Change);

end;

procedure TMain.btnMaleOfferClick(Sender: TObject);

var

Answer, I: Integer;

MainNode, CurrNode: PNode;

begin

// Check if an item is selected in the cbMale combo

// box.

if cbMale.ItemIndex > -1 then

begin

// Retrieve the data of the selected male node.

MainNode := MaleList.FindNodeByName(lvMale.

Selected.Caption);

// Retrieve the data of the female node selected in

// the cbMale combo box.

CurrNode := FemaleList.FindNodeByName(cbMale.Items.

Strings[cbMale.ItemIndex]);

// Display a message dialog showing information about

// the male and female nodes.

Answer := MessageDlg(CurrNode.Data.Name + Output +

sLineBreak +

GetNodeDataDescription(MainNode.Data),

mtInformation, [mbYes, mbNo], 0);

case Answer of

// If the user chooses 'Yes', indicating

// acceptance:

mrYes:

begin

// Remove the selected male node from the

// MaleList.

MaleList.RemoveNode(MainNode);

// Delete the selected male item from the lvMale

// list view.

lvMale.Items.Delete(lvMale.Selected.Index);

// Iterate through the items in the lvFemale list

// view.

for I := 0 to lvFemale.Items.Count - 1 do

begin

// Check if the name of the current female item

// matches the name of the accepted female

// node.

if lvFemale.Items[I].Caption =

CurrNode.Data.Name then

begin

// Delete the current female item from the

// lvFemale list view.

lvFemale.Items.Delete(I);

Break;

end;

end;

// Remove the accepted female node from the

// FemaleList.

FemaleList.RemoveNode(CurrNode);

// Update the male list view.

Self.UpdateMaleList;

// Update the female list view.

Self.UpdateFemaleList;

end;

end;

end;

end;

procedure TMain.lvMaleChange(Sender: TObject; Item:

TListItem; Change: TItemChange);

var

I: Integer;

MainNode, CurrNode: TNodeData;

begin

// Clear the items in the cbMale combo box.

cbMale.Items.Clear;

// Check if there are items in both lvMale and lvFemale

// list

// views and an item is selected in lvMale.

if (lvMale.Items.Count > 0) and (lvFemale.Items.Count >

0) and (lvMale.Selected <> nil) then

begin

// Retrieve the data of the selected male node.

MainNode := MaleList.FindNodeByName(lvMale.Selected.

Caption).Data;

// Iterate through the items in the lvFemale list

// view.

for I := 0 to lvFemale.Items.Count - 1 do

begin

// Retrieve the data of the current female node.

CurrNode := FemaleList.FindNodeByName(lvFemale.

Items[I].Caption).Data;

// Check the compatibility between the selected

// male node and

// the current female node.

if CheckCompatibility(MainNode, CurrNode) then

cbMale.Items.Add(CurrNode.Name);

end;

end;

// Trigger the lvFemaleChange event to update the

// cbFemale combo box.

if Sender <> nil then

lvFemaleChange(nil, Item, Change);

end;

class function TMain.GetNodeDataDescription(const

NodeData: TNodeData): string;

begin

// Build a string description of the NodeData.

Result := 'Name: ' + NodeData.Name + sLineBreak +

'Age: ' + IntToStr(NodeData.Age) + sLineBreak

+

'Height: ' + IntToStr(NodeData.Height) +

sLineBreak +

'Weight: ' + IntToStr(NodeData.Weight) +

sLineBreak +

'Habits: ' + NodeData.Habits + sLineBreak +

'Hobby: ' + NodeData.Hobby;

end;

procedure TMain.ImageClick(Sender: TObject);

begin

// Check if a sorting option is selected in cbSort.

if cbSort.ItemIndex > -1 then

begin

// Check the selected gender in cbSortGender.

case cbSortGender.ItemIndex of

0:

// If the selected gender is male and there are

// items in lvMale, sort the MaleList based on

// the selected sorting option.

if lvMale.Items.Count > 0 then

begin

MaleList.SortByParameter(SortingFunctions

[cbSort.ItemIndex]);

Self.UpdateMaleList;

end;

1:

// If the selected gender is female and there are

// items in lvFemale, sort the FemaleList based

// on the selected sorting option.

if lvFemale.Items.Count > 0 then

begin

FemaleList.SortByParameter(SortingFunctions

[cbSort.ItemIndex]);

Self.UpdateFemaleList;

end;

end;

end;

end;

procedure TMain.SaveDataToFile(Sender: TObject);

var

DataFile: file of TNodeData;

CurrentNode: PNode;

NodeData: TNodeData;

begin

// Check if the user selected a file to save the data.

if SaveDialog.Execute then

begin

// Assign the file specified in the SaveDialog to the

// DataFile variable.

AssignFile(DataFile, SaveDialog.FileName);

// Open the DataFile for writing, creating a new file

// or overwriting an existing one.

Rewrite(DataFile);

// Write the data from the MaleList to the file.

CurrentNode := MaleList.Head;

while CurrentNode <> nil do

begin

// Get the data from the current node.

NodeData := CurrentNode^.Data;

// Write the NodeData to the DataFile.

Write(DataFile, NodeData);

// Move to the next node in the MaleList.

CurrentNode := CurrentNode^.Next;

end;

// Write the data from the FemaleList to the file.

CurrentNode := FemaleList.Head;

while CurrentNode <> nil do

begin

// Get the data from the current node.

NodeData := CurrentNode^.Data;

// Write the NodeData to the DataFile.

Write(DataFile, NodeData);

// Move to the next node in the FemaleList.

CurrentNode := CurrentNode^.Next;

end;

// Close the DataFile after writing the data.

CloseFile(DataFile);

end;

end;

procedure TMain.LoadDataFromFile(Sender: TObject);

var

DataFile: file of TNodeData;

NodeData: TNodeData;

begin

// Check if the user selected a file to load the data

// from.

if SaveDialog.Execute then

begin

// Assign the file specified in the SaveDialog to the

// DataFile variable.

AssignFile(DataFile, SaveDialog.FileName);

// Open the DataFile for reading.

Reset(DataFile);

// Clear the MaleList and FemaleList to remove any

// existing data.

MaleList.Clear;

FemaleList.Clear;

// Read data from the DataFile until the end of the

// file is reached.

while not Eof(DataFile) do

begin

// Read a NodeData from the DataFile.

Read(DataFile, NodeData);

// Determine whether the NodeData belongs to a male

// or female based on the 'isMale' flag.

case NodeData.isMale of

True:

begin

// Add the NodeData to the MaleList.

MaleList.AddNode(NodeData);

// Add an item to the lvMale ListView with the

// NodeData's name.

lvMale.AddItem(NodeData.Name, nil);

end;

False:

begin

// Add the NodeData to the FemaleList.

FemaleList.AddNode(NodeData);

// Add an item to the lvFemale ListView with

// the NodeData's name.

lvFemale.AddItem(NodeData.Name, nil);

end;

end;

end;

// Close the DataFile after reading the data.

CloseFile(DataFile);

end;

end;

end.

Приложение Б

(обязательное)

Текст программного модуля frmInfo

unit frmInfo;

interface

uses

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

System.Variants, System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls,

uTypes;

type

TInfo = class(TForm)

lbName: TLabel;

lbAge: TLabel;

lbHeight: TLabel;

lbWeight: TLabel;

lbHabits: TLabel;

lbHobby: TLabel;

lbOptions: TLabel;

lbPreference: TLabel;

lbMinAge: TLabel;

lbMaxAge: TLabel;

lbMinHeight: TLabel;

lbMaxHeight: TLabel;

lbMinWeight: TLabel;

lbMaxWeight: TLabel;

edtName: TEdit;

edtAge: TEdit;

edtHeight: TEdit;

edtWeight: TEdit;

edtHabits: TEdit;

edtHobby: TEdit;

edtMinAge: TEdit;

edtMaxAge: TEdit;

edtMinHeight: TEdit;

edtMaxHeight: TEdit;

edtMinWeight: TEdit;

edtMaxWeight: TEdit;

btnOK: TButton;

btnCancel: TButton;

cbGender: TComboBox;

lbGender: TLabel;

procedure FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);

procedure FormShow(Sender: TObject);

private const

InvalidInput = 'Invalid input. Please enter valid data.';

public

{ Public declarations }

function TryGetNodeData(var ANodeData: TNodeData): Boolean;

end;

var

Info: TInfo;

implementation

{$R \*.dfm}

procedure TInfo.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);

begin

// Check if the Escape key was pressed.

if Key = VK\_ESCAPE then

// Set the modal result of the form to mrCancel, indicating that the form should be closed with a cancel result.

ModalResult := mrCancel

// Check if the Return key was pressed without the Shift key being held.

else if (Key = VK\_RETURN) and not (ssShift in Shift) then

// Set the modal result of the form to mrOk, indicating that the form should be closed with an OK result.

ModalResult := mrOk;

end;

procedure TInfo.FormShow(Sender: TObject);

begin

// Center the form on the screen.

Left := (Screen.Width - Width) shr 1;

Top := (Screen.Height - Height) shr 1;

// Set the focus to the edtName control, allowing the user to start typing immediately.

edtName.SetFocus;

end;

function TInfo.TryGetNodeData(var ANodeData: TNodeData): Boolean;

var

Age, Height, Weight: Integer;

minAge, maxAge, minHeight, maxHeight, minWeight, maxWeight: Integer;

begin

// Populate the form fields with the values from

// ANodeData

edtName.Text := ANodeData.Name;

edtHabits.Text:= ANodeData.Habits;

edtHobby.Text:= ANodeData.Hobby;

cbGender.ItemIndex := Ord(ANodeData.isMale);

edtAge.Text:= IntToStr(ANodeData.Age);

edtHeight.Text:= IntToStr(ANodeData.Height);

edtWeight.Text:= IntToStr(ANodeData.Weight);

edtMinAge.Text:= IntToStr(ANodeData.Preference.minAge);

edtMaxAge.Text:= IntToStr(ANodeData.Preference.maxAge);

edtMinHeight.Text:= IntToStr(ANodeData.Preference.

minHeight);

edtMaxHeight.Text:= IntToStr(ANodeData.Preference.

maxHeight);

edtMinWeight.Text:= IntToStr(ANodeData.Preference.

minWeight);

edtMaxWeight.Text:= IntToStr(ANodeData.Preference.

maxWeight);

repeat

ShowModal;

if ModalResult = MrOk then

begin

Result:= True;

// Try to convert the input values to integers

if TryStrToInt(edtAge.Text, Age) and

TryStrToInt(edtHeight.Text, Height) and

TryStrToInt(edtWeight.Text, Weight) and

TryStrToInt(edtMinAge.Text, minAge) and

TryStrToInt(edtMaxAge.Text, maxAge) and

TryStrToInt(edtMinHeight.Text, minHeight) and

TryStrToInt(edtMaxHeight.Text, maxHeight) and

TryStrToInt(edtMinWeight.Text, minWeight) and

TryStrToInt(edtMaxWeight.Text, maxWeight) then

begin

// Update ANodeData with the validated input

// values

ANodeData.Age := Age;

ANodeData.Height := Height;

ANodeData.Weight := Weight;

ANodeData.Preference.minAge := minAge;

ANodeData.Preference.maxAge := maxAge;

ANodeData.Preference.minHeight := minHeight;

ANodeData.Preference.maxHeight := maxHeight;

ANodeData.Preference.minWeight := minWeight;

ANodeData.Preference.maxWeight := maxWeight;

ANodeData.Name := edtName.Text;

ANodeData.Habits := edtHabits.Text;

ANodeData.Hobby := edtHobby.Text;

ANodeData.isMale := Boolean(cbGender.ItemIndex);

end

else

begin

// If the input values are invalid, show an error

// message and retry

ModalResult := mrRetry;

ShowMessage(InvalidInput);

end;

end

else if ModalResult = mrCancel then

Result:= False;

until ModalResult <> mrRetry;

end;

end.

Приложение В

(обязательное)

Текст программного модуля uTypes

unit uTypes;

interface

uses

System.SysUtils;

type

// Represents the preferences for a node

TPreference = record

minAge, maxAge: Integer;

minHeight, maxHeight: Integer;

minWeight, maxWeight: Integer;

end;

// Represents the data stored in each node of the linked list

TNodeData = record

SerialNum: Integer;

Name: ShortString;

Age: Integer;

Height: Integer;

Weight: Integer;

Habits: ShortString;

Hobby: ShortString;

isMale: Boolean;

Preference: TPreference;

end;

// Pointer to a node in the linked list

PNode = ^TNode;

// Represents a node in the linked list

TNode = record

Data: TNodeData;

Prev: PNode;

Next: PNode;

end;

// Function type for comparing two TNodeData instances

TCompareFunction = function(const AFirst, ASecond: TNodeData): Boolean;

// Doubly linked list class

TDoublyLinkedList = class

private

FHead: PNode;

FTail: PNode;

function CreateNode(const AData: TNodeData): PNode;

public

constructor Create;

destructor Destroy; override;

property Head: PNode read FHead;

procedure AddNode(const AData: TNodeData);

procedure RemoveNode(const ANode: PNode);

procedure Clear;

function FindNodeByName(const AName: string): PNode;

procedure SortByParameter(Compare: TCompareFunction);

end;

// Comparison functions for sorting the linked list based on different criteria

function CompareBySerialNumAsc(const AFirst, ASecond: TNodeData): Boolean;

function CompareBySerialNumDesc(const AFirst, ASecond: TNodeData): Boolean;

function CompareByNameAsc(const AFirst, ASecond: TNodeData): Boolean;

function CompareByNameDesc(const AFirst, ASecond: TNodeData): Boolean;

function CompareByAgeAsc(const AFirst, ASecond: TNodeData): Boolean;

function CompareByAgeDesc(const AFirst, ASecond: TNodeData): Boolean;

function CompareByHeightAsc(const AFirst, ASecond: TNodeData): Boolean;

function CompareByHeightDesc(const AFirst, ASecond: TNodeData): Boolean;

function CompareByWeightAsc(const AFirst, ASecond: TNodeData): Boolean;

function CompareByWeightDesc(const AFirst, ASecond: TNodeData): Boolean;

function CompareByHabitsAsc(const AFirst, ASecond: TNodeData): Boolean;

function CompareByHabitsDesc(const AFirst, ASecond: TNodeData): Boolean;

function CompareByHobbyAsc(const AFirst, ASecond: TNodeData): Boolean;

function CompareByHobbyDesc(const AFirst, ASecond: TNodeData): Boolean;

implementation

constructor TDoublyLinkedList.Create;

begin

inherited;

FHead := nil;

FTail := nil;

end;

destructor TDoublyLinkedList.Destroy;

begin

Clear;

inherited;

end;

procedure TDoublyLinkedList.SortByParameter(Compare: TCompareFunction);

var

Current, Running, Support: PNode;

TempData: TNodeData;

Swapped: Boolean;

begin

// Start with the head node

Current := FHead;

repeat

// Initialize variables

Swapped := False;

Support := Current;

Running := Current.Next;

while Running <> nil do

begin

if Compare(Running.Data, Support.Data) then

// Update the support node if the running node's data should come before

Support := Running;

// Move to the next node

Running := Running.Next;

end;

if Support <> Current then

begin

// Swap the data between the current node and the support node

TempData := Current.Data;

Current.Data := Support.Data;

Support.Data := TempData;

// Set the swapped flag to true

Swapped := True;

end;

// Move to the next node

Current := Current.Next;

// Repeat until no more swaps are made

until not Swapped;

end;

function TDoublyLinkedList.CreateNode(const AData: TNodeData): PNode;

begin

// Allocate memory for a new node

New(Result);

// Set the data of the new node

Result^.Data := AData;

// Initialize the previous pointer as nil

Result^.Prev := nil;

// Initialize the next pointer as nil

Result^.Next := nil;

end;

procedure TDoublyLinkedList.AddNode(const AData: TNodeData);

var

NewNode: PNode;

begin

// Create a new node with the given data

NewNode := CreateNode(AData);

if FHead = nil then

begin

// If the list is empty, set the new node as both the head and tail

FHead := NewNode;

FTail := NewNode;

end

else

begin

// Append the new node after the current tail

FTail^.Next := NewNode;

// Update the previous pointer of the new node

NewNode^.Prev := FTail;

// Update the tail to the new node

FTail := NewNode;

end;

end;

procedure TDoublyLinkedList.RemoveNode(const ANode: PNode);

begin

if ANode = FHead then

// Update the head if the node to be removed is the head

FHead := ANode^.Next;

if ANode = FTail then

// Update the tail if the node to be removed is the tail

FTail := ANode^.Prev;

if ANode^.Prev <> nil then

// Update the next pointer of the previous node

ANode^.Prev^.Next := ANode^.Next;

if ANode^.Next <> nil then

// Update the previous pointer of the next node

ANode^.Next^.Prev := ANode^.Prev;

// Deallocate the memory occupied by the removed node

Dispose(ANode);

end;

procedure TDoublyLinkedList.Clear;

var

CurrentNode: PNode;

NextNode: PNode;

begin

CurrentNode := FHead;

while CurrentNode <> nil do

begin

NextNode := CurrentNode^.Next;

// Deallocates the memory occupied by the current node

Dispose(CurrentNode);

CurrentNode := NextNode;

end;

// Resets the head (first) node to nil

FHead := nil;

// Resets the tail (last) node to nil

FTail := nil;

end;

function TDoublyLinkedList.FindNodeByName(const AName: string): PNode;

var

CurrentNode: PNode;

begin

// Initialize the result to nil

Result := nil;

// Start searching from the head (first) node

CurrentNode := FHead;

while CurrentNode <> nil do

begin

// Check if the name of the current node matches the desired name

if CurrentNode^.Data.Name = AName then

begin

// Store the reference to the current node as the result

Result := CurrentNode;

// Exit the loop since the node has been found

Break;

end;

// Move to the next node in the list

CurrentNode := CurrentNode^.Next;

end;

end;

function CompareBySerialNumAsc(const AFirst, ASecond: TNodeData): Boolean;

begin

Result := AFirst.SerialNum < ASecond.SerialNum;

end;

function CompareBySerialNumDesc(const AFirst, ASecond: TNodeData): Boolean;

begin

Result := AFirst.SerialNum > ASecond.SerialNum;

end;

function CompareByNameAsc(const AFirst, ASecond: TNodeData): Boolean;

begin

Result := AFirst.Name < ASecond.Name;

end;

function CompareByNameDesc(const AFirst, ASecond: TNodeData): Boolean;

begin

Result := AFirst.Name > ASecond.Name;

end;

function CompareByAgeAsc(const AFirst, ASecond: TNodeData): Boolean;

begin

Result := AFirst.Age < ASecond.Age;

end;

function CompareByAgeDesc(const AFirst, ASecond: TNodeData): Boolean;

begin

Result := AFirst.Age > ASecond.Age;

end;

function CompareByHeightAsc(const AFirst, ASecond: TNodeData): Boolean;

begin

Result := AFirst.Height < ASecond.Height;

end;

function CompareByHeightDesc(const AFirst, ASecond: TNodeData): Boolean;

begin

Result := AFirst.Height > ASecond.Height;

end;

function CompareByWeightAsc(const AFirst, ASecond: TNodeData): Boolean;

begin

Result := AFirst.Weight < ASecond.Weight;

end;

function CompareByWeightDesc(const AFirst, ASecond: TNodeData): Boolean;

begin

Result := AFirst.Weight > ASecond.Weight;

end;

function CompareByHabitsAsc(const AFirst, ASecond: TNodeData): Boolean;

begin

Result := AFirst.Habits < ASecond.Habits;

end;

function CompareByHabitsDesc(const AFirst, ASecond: TNodeData): Boolean;

begin

Result := AFirst.Habits > ASecond.Habits;

end;

function CompareByHobbyAsc(const AFirst, ASecond: TNodeData): Boolean;

begin

Result := AFirst.Hobby < ASecond.Hobby;

end;

function CompareByHobbyDesc(const AFirst, ASecond: TNodeData): Boolean;

begin

Result := AFirst.Hobby > ASecond.Hobby;

end;

end.