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 курсовой работе

на тему

ПРОГРАММНОЕ СРЕДСТО УЧЕТА РАБОТ САДОВОДА

БГУИР КР 1-40-01-01-110 ПЗ

Студент:  гр. 351001

        Орлович Д.С.

Руководитель:

асс. Фадеева Е.Е.

Минск 2024

**СОДЕРЖАНИЕ**

[**1 АНАЛИЗ ПРИМЕРОВ ИСПОЛЬЗОВАНИЯ ИНДЕКСНО-ПОСЛЕДОВАТЕЛЬНОГО МЕТОДА СРЕДИ ИЗВЕСТНЫХ РЕСУРСОВ И ТРЕБОВАНИЯ К ПРОЕКТИРУЕМОМУ ПРОГРАММНОМУ СРЕДСТВУ (ПС)** 7](#_Toc167546244)

[**1.1 Назначение разработки** 7](#_Toc167546245)

[**1.2 Состав выполняемых функций** 7](#_Toc167546246)

[**1.3 Входные и выходные данные** 7](#_Toc167546247)

[**2** **АНАЛИЗ ТРЕБОВАНИЙ К ПС И** **РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ** 8](#_Toc167546248)

[**2.1 Описание функциональности системы** 8](#_Toc167546249)

[**2.2 Спецификация функциональных требований** 8](#_Toc167546250)

[**2.2.1 Функция чтения файла с информацией о грядках:** 8](#_Toc167546251)

[**2.2.2 Функция просмотра сада:** 9](#_Toc167546252)

[**2.2.3 Функция выделения элементов сада:** 9](#_Toc167546253)

[**2.2.4 Функция удаления записи грядки:** 9](#_Toc167546254)

[**2.2.5 Функция удаления записи грядки:** 9](#_Toc167546255)

[**2.2.6 Функция добавления и удаления заметок о планировании работ:** 9](#_Toc167546256)

[**2.2.7 Функция масштабирования садового участка:** 10](#_Toc167546257)

[**2.2.8 Функция сохранения изменений:** 10](#_Toc167546258)

[**3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА** 11](#_Toc167546259)

[**3.1 Проектирование основных структур данных** 11](#_Toc167546260)

[**3.1.1 Структура данных о дачном участке** 11](#_Toc167546261)

[**3.1.2 Структура данных о цвете грядки** 11](#_Toc167546263)

[**3.1.3 Структура данных о списке культур** 11](#_Toc167546264)

[**3.1.4 Структура данных о списке грядок** 11](#_Toc167546265)

[**3.1.5 Структура данных о списке работ** 12](#_Toc167546266)

[**3.2 Проектирование алгоритмов работы динамическими структурами** 12](#_Toc167546267)

[**3.2.1 Алгоритм отображения дачного участка** 12](#_Toc167546268)

[**3.2.2 Алгоритм выбора грядки в просмотре сада** 12](#_Toc167546269)

[**3.2.3 Алгоритм выделения элементов массива садового участка** 12](#_Toc167546270)

[**4 СОЗДАНИЕ ПРОГРАММНОГО СРЕДСТВА** 14](#_Toc167546271)

[**4.1 Проектирование модулей программного средства** 14](#_Toc167546272)

[**4.2 Разработка модулей программного средства** 14](#_Toc167546273)

[**5** **ТЕСТИРОВАНИЕ, ПРОВЕРКА РАБОТОСПОСОБНОСТИ И АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ** 25](#_Toc167546274)

[**5.1 Тестирование программного средства** 25](#_Toc167546275)

[**5.2 Анализ полученных результатов** 27](#_Toc167546276)

[**6 РУКОВОДСТВО ПО УСТАНОВКЕ И ЭКСПЛУАТАЦИИ** 28](#_Toc167546277)

[**ЗАКЛЮЧЕНИЕ** 34](#_Toc167546278)

[**ПРИЛОЖЕНИЕ А** 35](#_Toc167546279)

[**ПРИЛОЖЕНИЕ В** 42](#_Toc167546280)

**ВВЕДЕНИЕ**

Желание, повысить уровень или объёмы производства возможно удовлетворить, при создании специального программного средства. Садоводы занимаются выращиванием овощей, фруктов, цветов и других растений на своих участках. Кроме того, стоит отметить, что выращивание растений имеет широкое общественное значение. Оно способствует экологической устойчивости, предоставляет людям свежие и качественные пищевые продукты, а также способствует физическому и эмоциональному благополучию отдельных садоводов.

Это самое программное средство по учёту работ садовода предназначено облегчить и ускорить работу с садовыми участками. Причиной создания программного связана с ростом размеров и сложности устройства садовых участков и соответствующая им сложность обслуживания этих участков.  Управление и учет работ, связанных с садоводством, может быть сложной задачей, особенно при увеличении размеров участка или числа выращиваемых культур.

Под дачным участком или садовым участком подразумевается, некоторая ограниченная территория, состоящая из грядок с растениями, которую в последствии возможно сделать другой грядкой с соответственно другим именем и другими растениями.

Цель данной курсовой работы состоит в изучении и создании программного средства для учета работ садовода. Предоставление программным средством необходимых инструментов служит для повышения эффективности и удобства управления собственным садовым участком и благодаря этому возможно достичь более качественных результатов в выращивании и уходом за растениями. Будут проанализированы различные способы взаимодействия с пользователем. Будут описаны необходимые алгоритмы и предложены их схемы, кроме того, будет проанализирована программа и корректность ее работы с реальными данными.

На основе требований к программному средству будет разработана архитектура программного средства и определены его функциональные возможности. В задачах так же стоит разработать принципы работы с данными, необходимыми для работы программы. И способы их хранения и доступа к ним.

# **1 АНАЛИЗ ПРИМЕРОВ ИСПОЛЬЗОВАНИЯ ИНДЕКСНО-ПОСЛЕДОВАТЕЛЬНОГО МЕТОДА СРЕДИ ИЗВЕСТНЫХ РЕСУРСОВ И ТРЕБОВАНИЯ К ПРОЕКТИРУЕМОМУ ПРОГРАММНОМУ СРЕДСТВУ (ПС)**

## **1.1 Назначение разработки**

Назначение разработки и цель курсовой работы: создать интуитивно-понятный принцип работы, интерфейс и реализовать, доступное для массового пользователя, программное средство для работы с дачным хозяйством. И свести к минимуму временные ресурсы пользователя.

Предоставить пользователю возможность добавление растения на определённую грядку. И добавить новую грядку, а так же поменять планировку садового участка.

Пользователю будет дана возможность так же вести журнал работ.

## **1.2 Состав выполняемых функций**

1. Чтение данных из файлов, хранящие различные данные пользователя;
2. Организация взаимодействия с пользователем;
3. Редактирование данных о дачном хозяйстве в процессе работы;
4. Возможность изменять масштаб дачного участка.
5. Сохранение изменённых данных.
6. Наглядная демонстрация пользователю количества растений на какой-либо грядке.
7. Возможность просмотра места, где растёт растение.
8. Добавление и удаление запланированных работ

## **1.3 Входные и выходные данные**

На входе: вводимые данные пользователем, файлы с типом о дачном хозяйстве;

На выходе: файлы с типом об обновлённом дачном хозяйстве.

# **2** **АНАЛИЗ ТРЕБОВАНИЙ К ПС И** **РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ**

## **2.1 Описание функциональности системы**

Работа с различными частями разрабатываемого программного средства должна быть предоставлена пользователю в комфортной форме.

При запуске программы сразу же начинается чтение из различных файлов данных о дачном участке пользователя. Дачный участок представлен в виде двумерного массива, каждый элемент которого хранит информацию о грядке и растении, которое растёт на грядке. Различные грядки помечаются разными цветами. Группа элементов массива дачного участка, имеющих общее имя грядки(одни цвет), считается грядкой, независимо от места их расположения. Файл с культурами(грядками) хранит не только культуры(грядки) в массиве дачного участка, а вообще все культуры(грядки) на участке. Файл с списком посадок, любая из которых имеет дату посадки, описание и название самой работы. В программе предполагается взаимодействие с только теми культурами(грядками), которые пользователь решил создать сам или оставить из предложенных при первом запуске.

При сохранение в файл с массивом садового участка в файл поступаютстолбцы массива.

Пользователь должен иметь в доступе функции для удаления какой-либо культуры или грядки и функции для добавления грядки или культуры. Для добавления нового элемента(культуры или грядки) необходимо соблюсти определённые правила. Функции внесения новых данных предполагает создание новой записи. Все ети функции должны быть доступны пользователю напрямую.

Завершение программы возможно при нажатии кнопки специально отведённой под эту задачу. Сохранение или несохранение данных определяется желание пользователя. Возможность сохранения доступна при нажатии определённой комбинации клавиш и кнопки на окне программы.

**2.2 Спецификация функциональных требований**

Функции программы должны быть реализованы и оформлены в соответствии со следующими требованиями:

**2.2.1 Функция чтения файла с информацией о грядках:**

1. Поиск нужного файла осуществляется по уникальному имени с расширением \*.cul.
2. Данная функция начинает выполняться при запуске программного обеспечения.
3. Цвет грядки определяется по коду грядки в элементе садового участка.
4. Чтение происходит по столбцам массива.
5. Размера садового участка(массива) постоянны.

**2.2.2 Функция просмотра сада:**

1. Следует иметь возможность демонстрации расположения культур на всём садовом участке.
2. Возможность выбора грядки доступна двумя способами.
3. По выбору грядки должен произойти подсчёт и вывод информации об культурах на этой грядке.
4. Это функция не может изменить данные об садовом участке а только использовать эти данные.
5. Выделение мест, где растёт культура не должно изменять цвета других мест отрисовки.
6. Выбор какой-либо грядки должен оставить ярко заметной только выбранную грядку.

**2.2.3 Функция выделения элементов сада:**

1. Происходит двумя методами (одиночное и множественное).
2. Множественное выделение должно происходить с верхнего левого угля в противоположный.
3. Множественное выделение происходит при удержании левой кнопки мыши.
4. При одиночном выборе окно редактирования уже содержит информацию об выделенном элементе.

**2.2.4 Функция удаления записи грядки:**

1. Удаление грядки доступно только при выборе грядки в окне изменения грядки.
2. Удаление разрешено при условии, что она не используется в данный момент.
3. После удаление грядки её имя и цвет становятся доступными для использования в создании новой грядки.

**2.2.5 Функция удаления записи грядки:**

1. Удаление культуры доступно только при выборе культуры в окне изменения грядки.
2. Удаление разрешено при условии, что она не используется в данный момент.
3. После удаление культуры её название становятся доступными для использования в создании новой грядки.

**2.2.6 Функция добавления и удаления заметок о планировании работ:**

1. Данная функция начинает своё выполнение после открытия формы представляющей все запланированные работы и после нажатия пользователем определённой кнопки
2. При добавлении обязательным условии является уникальное и непустое название работы
3. Функция удаления становится доступной при выборе какого-либо названия работы

**2.2.7 Функция масштабирования садового участка:**

1. При первом запуске программы масштаба максимально возможный;
2. Изменение масштаба возможно при нажатии двух кнопок;
3. Имеется верхний и нижний предел масштаба;

**2.2.8 Функция сохранения изменений:**

1. Данная функция начинает своё выполнение после нажатия пользователем определённой кнопки
2. Сохраняет изменения во все файлы.
3. После сохранения пользователю выводится сообщение об успешном сохранении.

**3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

**3.1 Проектирование основных структур данных**

**3.1.1 Структура данных о дачном участке**

Дачный участок - это просто двумерный массив, каждый элемент которого просто запись, включающая в себя запись культуры и номер грядки, к которой этот элемент относиться.

Элемент дачного участка содержит поля:

1. Код грядки
2. Запись о культуре

Код культуры устанавливается в файле, со всеми возможными кодами культуры. Каждому коду ставится в соответствие своё имя культуры.

Так же для удобства работы с грядками, к которым принадлежат различные элементы массива, было решено сделать отдельную запись под грядки, в которой хранится все имена грядок и их номера.

**3.1.2 Структура данных о цвете грядки**

 Для обеспечения функциональности сопоставления номера грядки с уникальным цветом в программном средстве, необходимо создать способ определения цвета грядки по её номеру.

Предлагается использовать структуру данных, по функционалу похожую на хэш-таблицу, с ключом и значением:

Номер грядки(ключ) – целочисленное значение, которое будет однозначно идентифицировать каждую грядку в саду.

Цвет грядки(значение) – уникальный цвет самой грядки

**3.1.3 Структура данных о списке культур**

Список культур - это динамическая структура данных, содержащая записи о культуре и указатель на следующую запись.

Запись культуры представляет из себя просто несколько полей:

1. Срок произрастания.
2. Уникальное имя культуры.
3. Уникальный код культуры.

**3.1.4 Структура данных о списке грядок**

Список грядок - это динамическая структура данных, содержащая записи о грядке и указатель на следующую запись.

Запись грядки представляет из себя просто несколько полей:

1. Уникальный код грядки.
2. Уникальное имя грядки.

**3.1.5 Структура данных о списке работ**

Список работ - это динамическая структура данных, содержащая записи о работе и указатель на следующую запись.

Запись работ представляет из себя просто несколько полей:

1. Уникальный название работы.
2. Какое угодно описание.
3. Какую угодно запланированную дату.

Чтобы не ограничивать пользователя в формате даты выполнения работы, то поле отвечающее за дату выполнения это лишь строка, в которую пользователь может записать любой текст. Предполагается, что пользователь введёт строку даты по типу: “11 апреля”, ”после вторника, перед авто-лавкой”.

**3.2 Проектирование алгоритмов работы динамическими структурами**

**3.2.1 Алгоритм отображения дачного участка**

Изначально для определения области, в которой будет происходить отрисовка дачного участка, определяется размер каждого элемента дачного участка и их количества.

Для более удобного вывода дачного участка отрисовка происходит в центре области для отрисовки, которая меняет при изменении масштаба.

При изменении масштаба происходит новый расчёт центра отрисовки и размера элементов садового участка.

При выборе пользователем элементов цвет выделения зависит от цвета выделяемого элемента, который определяется координатой нажатия мыши.

**3.2.2 Алгоритм выбора грядки в просмотре сада**

Выбор грядки возможен выбором её в области отрисовки или выбором из списка. Выбирая грядку из списка мы узнаём сначала её имя а потом уже и индекс, а при выборе грядки через область отрисовки мы узнаём сразу её индекс.

Как только известно индекс грядки происходит перебор всего массива и подсчёт количества культур при условии совпадения индекса проверяемой и выбранной грядки. Подсчёт реализуется с использованием структуры данных реализующей функционал хеш-таблицы.

Отрисовка происходит также при переборе всего массива и при условии совпадения индексов.

**3.2.3 Алгоритм выделения элементов массива садового участка**

Как только происходит нажатие на ЛКМ то инициализируется начальная точка отрисовка хранящая индексы элемента массива.

Как только курсор начинает своё движение по области отрисовки, то происходит определение текущего элемента, а дальнейшее определяется состояние ЛКМ:

1. Если ЛКМ зажата, то происходит отрисовка всех элементов от начального и до конечного.
2. Ули ЛКМ не зажата то происходит отрисовка только текущего элемента.

**4 СОЗДАНИЕ ПРОГРАММНОГО СРЕДСТВА**

**4.1 Проектирование модулей программного средства**

Программное средство разрабатывается на основе спроектированных алгоритмов (см. раздел 3) и спецификации функциональных требований (см. раздел 2). Программное средство имеет модули:

1. uMain – модуль основной формы программы;
2. uMain\_logical – вспомогательный модуль основной формы выполняющий работу с файлами.
3. uGarden – модуль формы об изменениях грядки;
4. uGarden\_logical – вспомогательный модуль изменения грядки, выполняющий проверки и изменения данных о садовом участке;
5. uAddNewCulture – модуль формы добавления культуры;
6. uAddNewGarden – модуль формы добавления грядки;
7. uShowAllGardens – модуль формы просмотра садового участка;
8. uDictionary – модуль с вспомогательной структурой данных;
9. uGlobalData – модуль с функциями и переменными участвующими в большей части программы;
10. uSP – модуль формы о справке программного средства;
11. uWorks– модуль формы, демонстрирующий все записи о запланированных работ;
12. uAddWork – модуль формы, выполняющий функции добавление новой записи работе;

Код программы см. Приложение В.

**4.2 Разработка модулей программного средства**

Таблица 4.1 – Описание подпрограмм.

|  |  |  |
| --- | --- | --- |
| Имя и тип под- программы | Назначение подпрограммы | Параметры, назначение и возвращаемые значения |
| Procedure FormCreate | Чтение всех данных из файлов, инициализация основный структур данных | (Sender: TObject);  Sender – объект работы процедуры (форма) |

Продолжение – таблица 4.1

|  |  |  |
| --- | --- | --- |
| Procedure PaintBox1MouseDown | Определение начального элемента выделения грядок | (Sender: TObject; Button: TMouseButton;  Shift: TShiftState; x, y: integer)  Sender – объект работы процедуры (область для рисования)  Shift – состояние клавиши Shift на текущий момент  X,Y – положение курсора |
| Procedure PaintBox1Click | Открытие формы изменения грядок | (Sender: TObject);  Sender – объект работы процедуры (область для рисования) |
| Procedure PaintBox1MouseLeave | Стирание последнего выделения на садовом участке | (Sender: TObject);  Sender – объект работы процедуры (область для рисования) |
| Procedure PaintBox1MouseMove | Отрисовка выделенных грядок и определение текущего(крайнего элемента выделения) | (Sender: TObject;  Shift: TShiftState; x, y: integer)  Sender – объект работы процедуры (область для рисования)  Shift – состояние клавиши Shift на момент возникновения события  X,Y – положение курсора |
| Procedure PaintBox1Paint | Отрисовка всего дачного участка | (Sender: TObject);  Sender – объект работы процедуры (область для рисования) |
| Procedure ActionMinusExecute | Уменьшение масштаба дачного хохзяйства | (Sender: TObject);  Sender – объект работы процедуры (список событий формы) |

Продолжение – таблица 4.1

|  |  |  |
| --- | --- | --- |
| Procedure ActionPlusExecute | Увеличение масштаба дачного хозяйства | (Sender: TObject);  Sender – объект работы процедуры (список событий формы |
| Procedure ActionShowAllExecute | Отображение формы показа дачного хозяйства | (Sender: TObject);  Sender – объект работы процедуры (список событий формы |
| Procedure ActionSaveExecute | Сохранение всех данных в файлы | (Sender: TObject);  Sender – объект работы процедуры (список событий формы |
| Procedure FormShow | При отображении основной формы запускается передача в форму изменения грядок определённые структуры | (Sender: TObject);  Sender – объект работы процедуры (форма) |
| Procedure createLists | Инициализирует необходимые для работы формы списки | (Culturelist: PtCulture; Gardenlist: PtGarden);  Culturelist – динамический список культур  Gardenlist – динамиеский список грядок |
| Procedure ActionAddCultureExecute | Отображение формы добавления культуры. Добавление происходит в зависимости от итога закрытия формы добавления | (Sender: TObject);  Sender – объект работы процедуры (список событий формы) |

Продолжение – таблица 4.1

|  |  |  |
| --- | --- | --- |
| Procedure ActionAddGardenExecute | Отображение формы добавления грядки. Добавление происходит в зависимости от итога закрытия формы добавления | (Sender: TObject);  Sender – объект работы процедуры (список событий формы) |
| Procedure ActionDeleteCultureExecute | Удаление записи о выбранной культуре | (Sender: TObject);  Sender – объект работы процедуры (список событий формы) |
| Procedure ActionDeleteGardenExecute | Удаление записи о выбранной грядке | (Sender: TObject);  Sender – объект работы процедуры (список событий формы) |
| Procedure ButtonSaveClick | Применение изменений грядки к массиву дачного участка | (Sender: TObject);  Sender – объект работы процедуры (кнопка) |
| function FormShowForChange | Редактирование только одной грядки | (CurrPoint: TPoint);  CurrPoint – точка хранящая индексы элемента дачного участка для редактирования |
| function FormShowForChangeCtrl | Редактирование только нескольких грядок | (CurrPoint, StartPoint: TPoint);  CurrPoint – начальная точка(верхняя левая) хранящая индексы элемента дачного участка для редактирования  StartPoint – конечная точка(нижняя правая) хранящая индексы элемента дачного участка для редактирования |

Продолжение – таблица 4.1

|  |  |  |
| --- | --- | --- |
| Procedure ListViewCultureClick | Определение выбранной записи культур | (Sender: TObject);  Sender – объект работы процедуры (список) |
| Procedure ListViewGardenClick | Определение выбранной записи грядок | (Sender: TObject);  Sender – объект работы процедуры (список) |
| Procedure CreateListViewGarden | Создание строк грядок в компоненте формы, отображающей все грядки | (list: PtGarden);  list – динамиеский список грядок |
| Procedure CreateListViewCulture | Создание строк культур в компоненте формы, отображающей все культуры | (list: PtCulture);  list – динамиеский список культур |
| function ShowForAdd | Инициализирует необходимые для работы формы списки и передача элемента для возможного добавления | (var culture: cultureListInfo; culturelist: ptculture);  Culture – добавляемая культура  culturelist– динамиеский список культур |
| function IsValidInputData | Проверка на корректность введённых данных культуры | (const strName, const strTime: string)  strName - имя добавляемой грядки  strTime – время созревания культуры, представленной в строке |

Продолжение – таблица 4.1

|  |  |  |
| --- | --- | --- |
| Procedure ButtonSaveClick | Сохранение данных культуры (элемент будет добавлен) | (Sender: TObject);  Sender – объект работы процедуры (кнопка) |
| procedure ButtonCancelClick | Не сохраняет данные о культуре (элемент не будет добавлен) | (Sender: TObject);  Sender – объект работы процедуры (кнопка) |
| procedure IsValidInputData | Инициализирует необходимые для работы формы списки и передача элемента для возможного добавления | (const strName: string; const color: TColor);  strName – имя добавляемой грядки  color – цвет добавляемой грядки |
| function ShowForAdd | Проверка на корректность введённых данных грядки | (var garden: TGARDEN; gardenlist: ptgarden)  garden – добавляемая грядка  gardenlist – динамиеский список грядок |
| procedure ButtonSaveClick | Сохранение данных грядки (элемент будет добавлен) | (Sender: TObject);  Sender – объект работы процедуры (кнопка) |
| procedure ButtonCancelClick | Не сохраняет данные о грядке (элемент не будет добавлен) | (Sender: TObject);  Sender – объект работы процедуры (кнопка) |
| Function FormShowAll | Инициализирует необходимые для работы формы списки | (culturelist: ptCulture; gardenlist: PtGarden);  Culturelist – динамический список культур  Gardenlist – динамиеский список грядок |

Продолжение – таблица 4.1

|  |  |  |
| --- | --- | --- |
| procedure FormShow | Начальное отображение на компоненте всех культур и грядок | (Sender: TObject);  Sender – объект работы процедуры (форма) |
| Procedure FormCreate | Создание нужного формата компонентов форм | (Sender: TObject);  Sender – объект работы процедуры (форма) |
| Procedure PrintSelectedGarden | Выводит выбранную из списка компонента формы грядку | (const idGarden: integer);  idGarden – индекс выдранной грядки |
| Procedure PaintBox1MouseDown | Выводит выбранную по нажатию грядку | (Sender: TObject; Button: TMouseButton; Shift: TShiftState; x, y: integer);  Sender – объект работы процедуры (область для рисования)  Shift – состояние клавиши Shift на момент возникновения события  X,Y – положение курсора |
| Procedure PrintSelectedCulture | Отображает культуры на дачном участке | (const nameCulture: string);  nameCulture – имя выбранной строки |
| Procedure CreateListCultureAll | Выводит на компонент формы все культуры на дачном участке | (CultutrList: ptCulture);  Culturelist – динамический список культур |
| Procedure CreateListViewGarden | Выводит на компонент формы все грядки на дачном участке | (gardenlist: PtGarden);  Gardenlist – динамиеский список грядок |
| Procedure CreateListViewCulture | Выводит на компонент формы все культуры на дачном участке принадлежащие какой-либо выбранной грядке | (const idGarden: array of integer);  idGarden – массив все грядок для отображения |
| Procedure printAllGarden | Отрисовывает весь садовый участок | - |
| procedure SaveFileGarden; | Запись в файл всех грядок в садовом участке | (list: PtGarden);  list – динамический список грядок |
| procedure ReadFileGarden; | Чтение из файла всех грядок в садовом участке | (list: PtGarden);  list – динамический список грядок |
| procedure ReadFileCulture; | Чтение из файла всех культур в садовом участке | (list: PtCulture);  list – динамический список культур |
| procedure SaveFileCulture; | Запись в файл всех культур в садовом участке | (list: PtCulture);  list – динамический список культур |
| procedure ReadFileColor; | Чтение файла с индексами грядок и их цвета | - |
| procedure SaveFileColor; | Запись в файл всех индексов грядок и из цветов | - |
| procedure SaveGardenMas; | Запись в файл массива дачного участка | - |

Продолжение – таблица 4.1

|  |  |  |
| --- | --- | --- |
| procedure ReadGardenMas; | Чтение из файла массива дачного участка | - |
| function CheckCultureUsing | Проверка на использование в садовом участке определённой культуры | (const culture: string);  Culture – имя культуры |
| function CheckGardenUsing | Проверка на использование в садовом участке определённой грядки | (const gardencod: integer);  Gardencod – индекс грядки |
| procedure DeleteCulture | Удаление из списка культур определённой записи | (const name: string; list: Ptculture);  Name – имя культуры для удалния  list – динамический список культур |
| procedure DeleteGarden | Удаление из списка грядок определённой записи | (const name: string; list: ptgarden);  Name – имягрядки для удалния  list – динамический список грядок |
| procedure AddCulture | Добавление в список культур записи | (const newCulture: cultureListInfo; list: Ptculture);  newCulture – имя культуры для добавления  list – динамический список культур |

Продолжение – таблица 4.1

|  |  |  |
| --- | --- | --- |
| procedure AddGarden | Добавление в список грядок записи | (const newgarden: TGarden; list: ptgarden);  newgarden – имя культуры для добавления  list – динамический список грядок |
| function IsValidGarden | Проверка на валидность введённого имени грядки | (list: ptgarden; const str: string; var cod: integer);  list – динамический список грядок  str – имя грядки  cod – индекс грядки в списке грядок |
| function IsValidCulture | Проверка на валидность введённого названия культуры | (list: Ptculture; const str: string; var cod: integer);  list – динамический список культур  str – имя культуры  cod – индекс культуры в списке культур |
| function IdentifyColor | Определение цвета грядки по её коду | (const cod: integer)  cod – индекс шрядки в списке грядок |
| function ConvertStringToDate | Конвертация из строки в дата-тип | (const dateStr: string; var date: TMyDate);  dateStr – строка хранящая дату  date – дата созданная из строки |

Продолжение – таблица 4.1

|  |  |  |
| --- | --- | --- |
| function ConvertDateToString | Конвертация из дата-типа в строку | (const dateStr: TMyDate)  dateStr – дата которая будет конвертированна в строку |
| function GetIdGarden | Возврат индекса грядки по её имени | (const Name: string; list: PtGarden);  Name – имя грядки  list – динамический список грядок |
| Procedure createListView1 | Заполнение компонента формы, демонстрирующий список всех запланированных работ | (Sender: TObject)  Sender – объект работы процедуры (область для записей) |
| procedure MyShow | Инициализация в модуле списка работ | (list: ptworks)  List - динамический список работ |
| Procedure ButtonOkClick | Подтверждение добавления записи о работе при условии корректности введённых данных о работе | (Sender: TObject)  Sender – объект работы процедуры (кнопка) |
| Function IsValidInputData | Проверка на правильность введённых данных о работе | (const str1: string)  str1 – строка с названием добавляемой работы |

**5** **ТЕСТИРОВАНИЕ, ПРОВЕРКА РАБОТОСПОСОБНОСТИ И АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ**

**5.1 Тестирование программного средства**

Таблица 5.1 – Тестирование программного средства

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Специфика тестирования | Номер теста | Вводимые данные | Ожидаемый результат | Полученный результат |
| Запуск программы | 1 | - | Отображение садового участка, все кнопки из меню должны иметь иконку | Тест пройден |
| Отображения садового участка | 2 | Нажатие кнопок увеличения/уменьшая масштаба | Увеличение / уменьшение клеток каждого элемента садового участка | Тест пройден |
| Добавление записи культуры | 3 | Нажатие кнопки добавления культуры, ввод данных и добавление | Открытие формы добавления культуры, при условии корректности введённых данных, изменение структуры данных культур | Тест пройден |
| Добавление записи грядки | 4 | Нажатие кнопки добавления грядки, ввод данных и добавление | Открытие формы добавления грядки, при условии корректности введённых данных, изменение структуры данных грядки | Тест пройден |
| Удаление записи культуры | 5 | Нажатие кнопки удаления культуры | Открытие формы удаления культуры, изменение структуры данных культур | Тест пройден |
| Удаление записи грядки | 6 | Нажатие кнопки удаления грядки | Открытие формы удаления грядки, изменение структуры данных грядки | Тест пройден |
| Выделение элемента сада | 7 | Движение курсора по области рисования | Выделение только одного элемента сада | Тест пройден |

Продолжение – таблица 5.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Выделение элементов сада | 8 | Движение курсора по области рисования с зажатой ЛКМ | Выделение в области прямоугольника сразу нескольких элементов сада | Тест пройден |
| Подготовка данных для редактирования грядок | 9 | Выделение элементов сада | Отображение формы со всеми списками культур и грядок | Тест пройден |
| Редактирование элементов сада (грядки) | 10 | Изменение цвета грядки | Цвет меняется для всех выделенных элементов сада | Тест пройден |
| Редактирование элементов сада (культур) | 11 | Изменение культуры | Культура изменяется для всех выделенных элементов сада | Тест пройден |
| Просмотр сада | 12 | Нажатие пункта открытия просмотра сада | Открытие формы с уже отрисованным дачным участком | Тест пройден |
| Отображение грядки | 13 | Нажатие определённой грядки в списке или в области отрисовки | Отображение только выделенной грядки | Тест пройден |
| Отображение культуры | 14 | Нажатие определённой культуры в списке | Выделение определённым образом только элементов с нужной культурой | Тест пройден |
| Счёт количества культур | 15 | Выбор определённой грядки | Отображение всех культур на всех элементах грядки и подсчёт их количества | Тест пройден |
| Добавление записи работы | 16 | Нажатие кнопки добавления работы, ввод данных и добавление | Открытие окна добавления записи о работе | Тест пройден |

Продолжение – таблица 5.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Удаление записи работы | 17 | Нажатие кнопки удаления культуры, при условии выделения какой-либо работы | Удаление из списка записи работы | Тест пройден |
| Выход из программы | 18 | Нажатие кнопки “крестик” | Завершение программы, закрытие формы | Тест пройден |
| Сохранение изменений | 19 | Нажатие определённых клавиш или кнопки сохранения | Сохранение всех данных | Тест пройден |
| Загрузка данных после их сохранения | 20 | Запуск программы после сохранения | Отображения сада в соответствии с последним сохранением | Тест пройден |

**5.2 Анализ полученных результатов**

Ошибок при тестировании не найдено. Программное средство работает корректно. Все формы отображают верные данные, в нужном порядке. Все события, создаваемые пользователем работают правильно. Все результаты взаимодействия пользователя происходят соответствуя ожиданию.

Структура данных культур и грядок по ходу выполнения программы хранят корректные данные. Все проверки на адекватность введённых данных выполняют свои функции.

Все формы программы появляются при ожидаемой необходимости и отображают компоненты в нужном месте и отображают верные данные. Время отклика программы не превышает критических значений.

**6 РУКОВОДСТВО ПО УСТАНОВКЕ И ЭКСПЛУАТАЦИИ**

1. Установите exe-файл программы в одной папке с типизированными файлами данных расширения \*.cul.

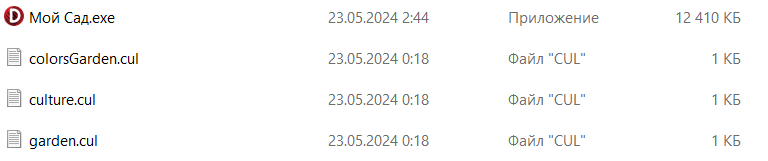


Рис. 6.1 – Пример расположения некоторых необходимых для работы файлов.

1. Основная форма программы

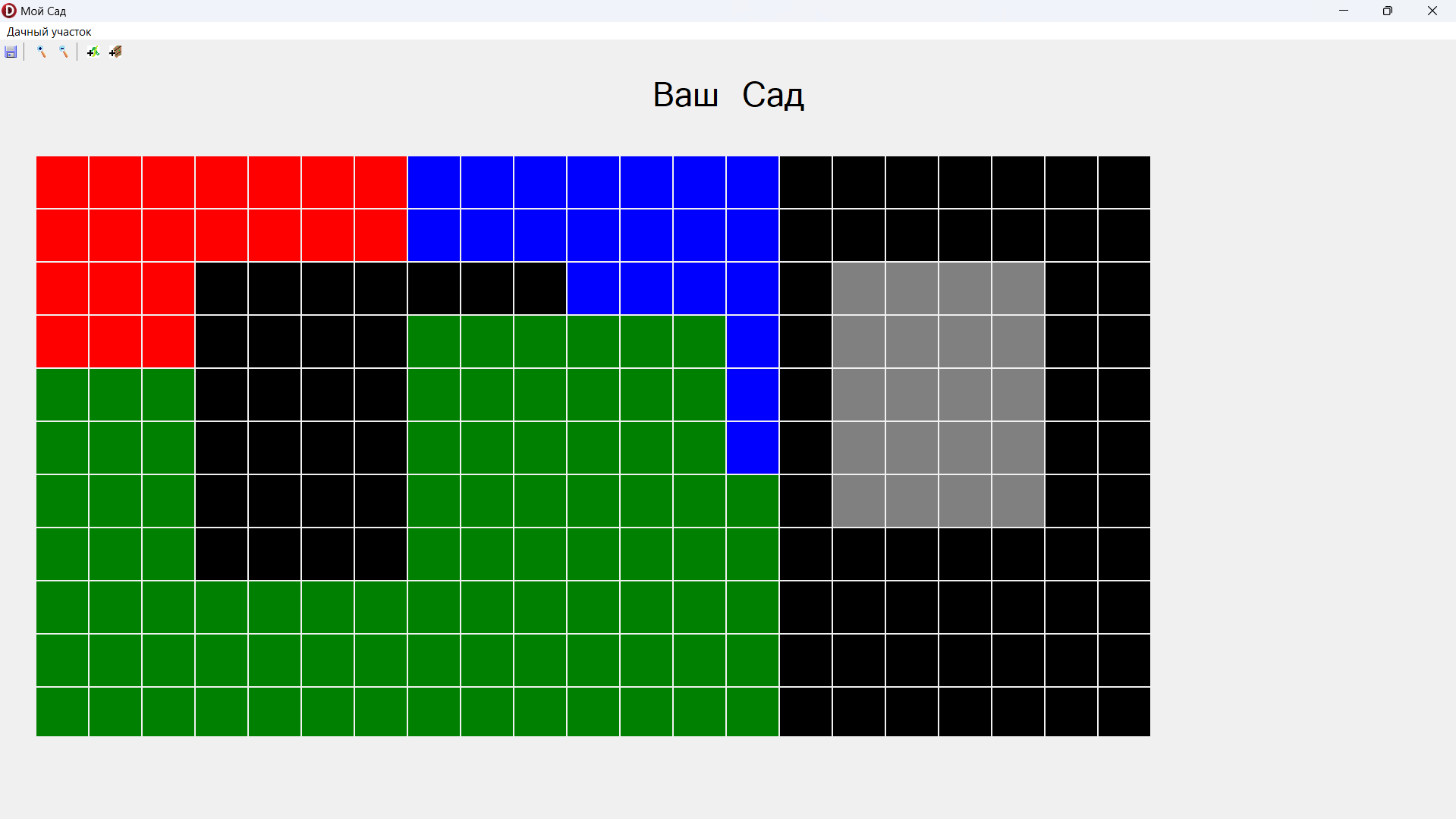


Рис. 6.2 – Основное меню программы.

1. Выделение нескольких элементов сада

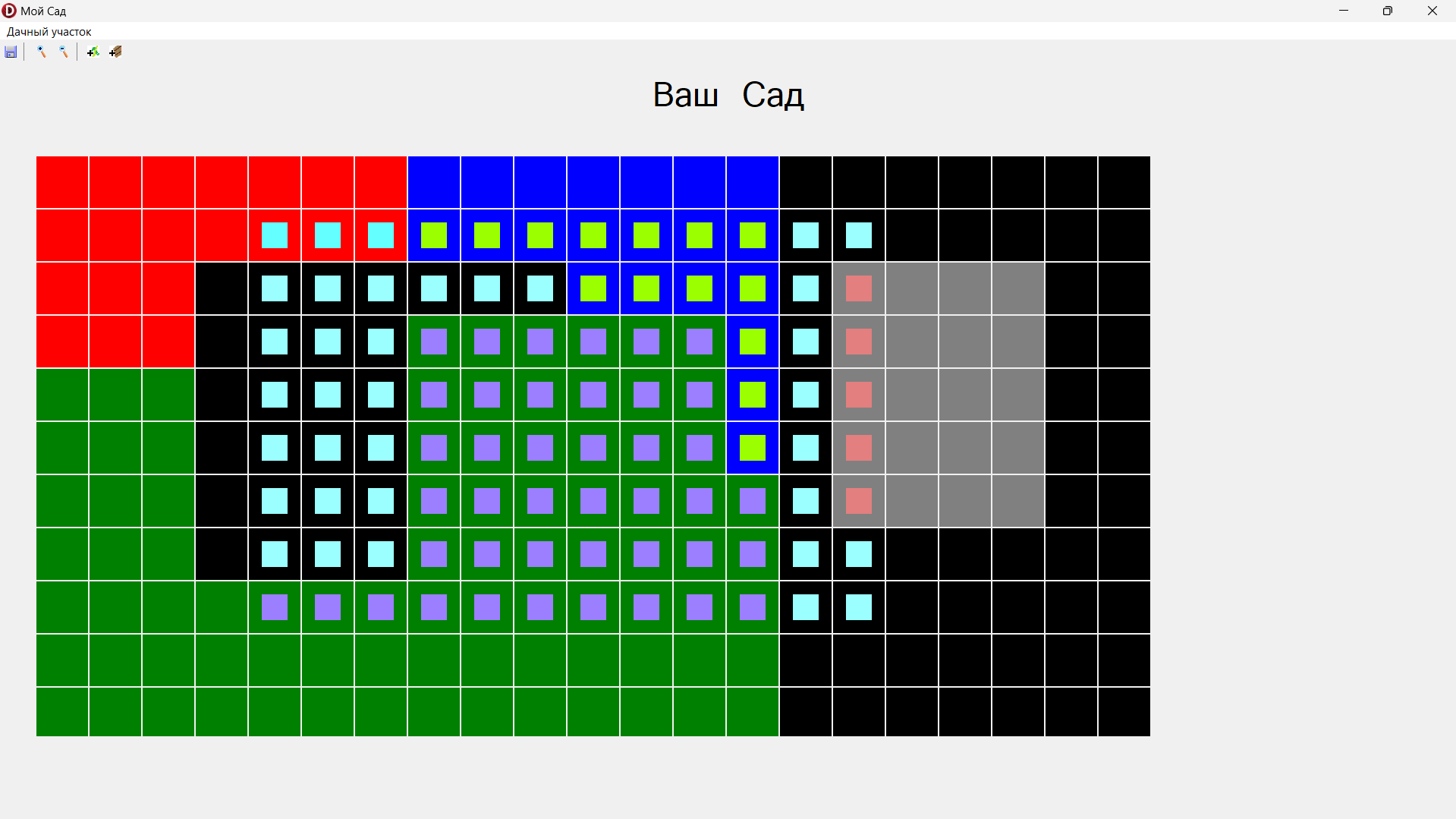


Рис. 6.3 – Выделение грядок для редактирования.

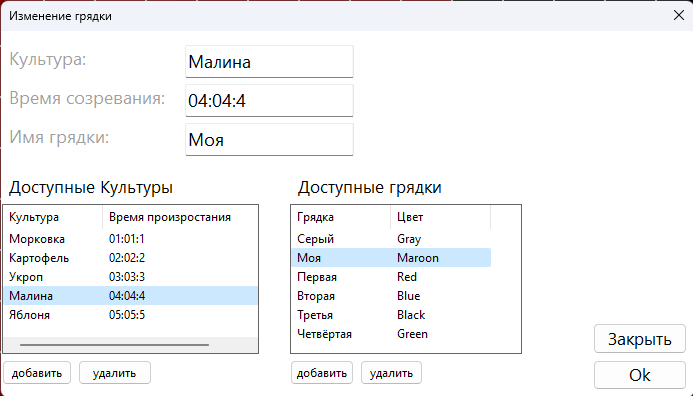
1. Редактирование грядок при выборе определённых элементов сада
2. 

Рис. 6.4 – Окно редактирования грядок.

1. При нажатии кнопки “Ок” выбранные изменения отобразятся на форме

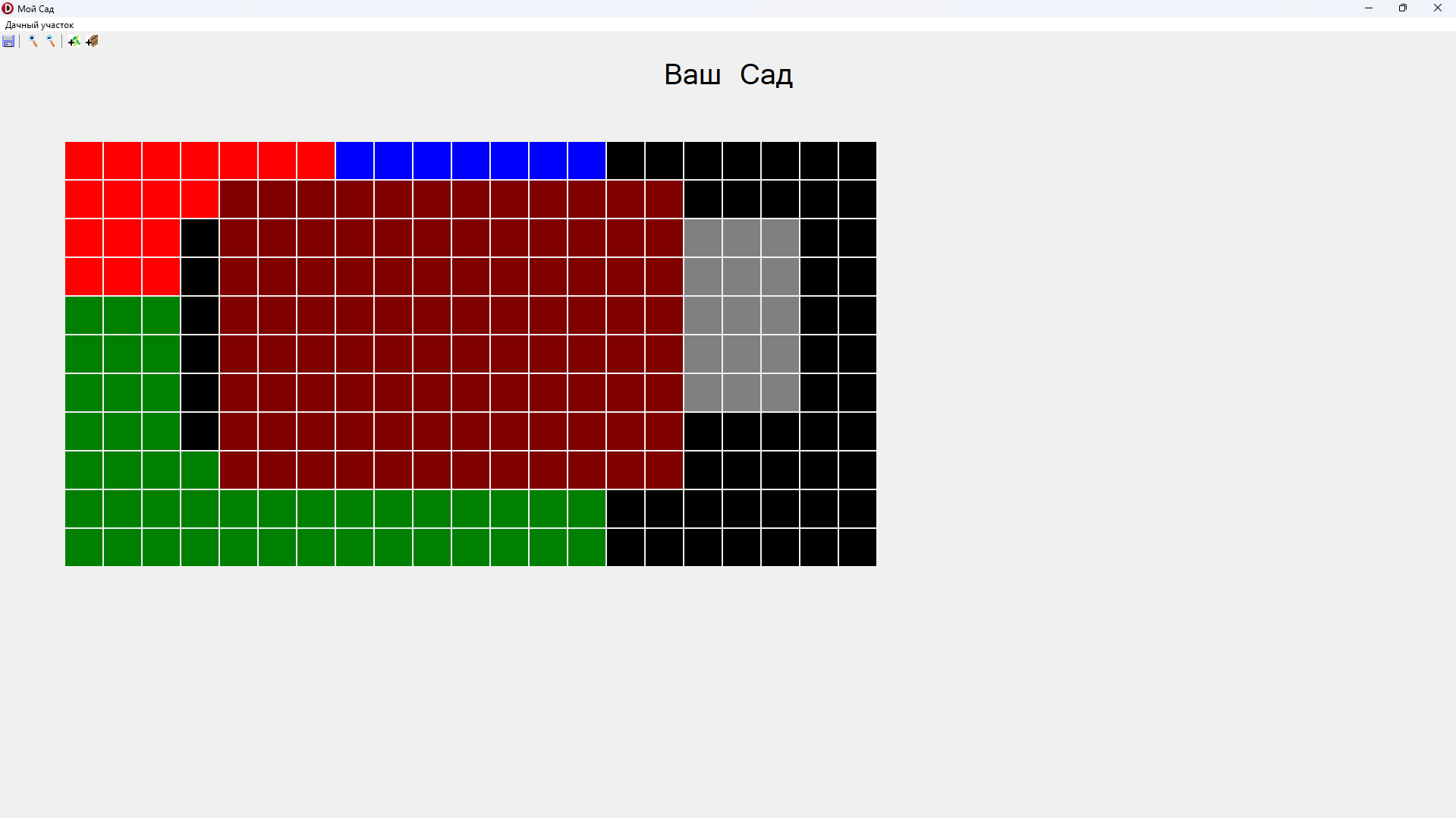


Рис. 6.5 – Основное меню программы.

1. При нажатии кнопки добавления грядки откроется окно добавления новой грядки

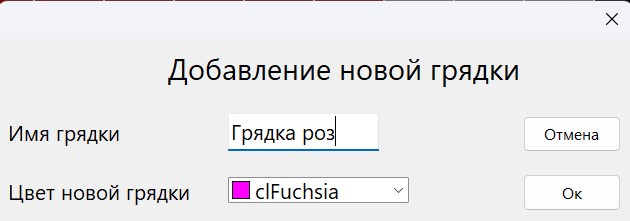


Рис. 6.6 – Форма добавления новой грядки

1. При попытке ввести недопустимые значения будет получено уведомление

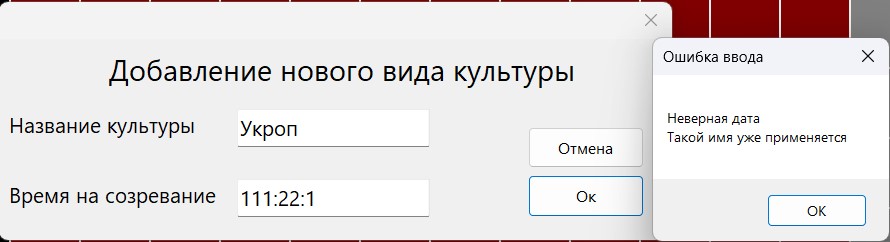


Рис. 6.7 – Информирование о некорректности введённых данных

1. При нажатии кнопки сохранения или комбинации клавиш Ctrl + S произойдёт сохранение всех данных

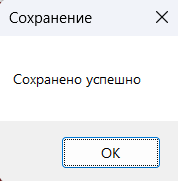


Рис. 6.8 – Информирование о сохранении данных

1. При нажатии раздела меню “Дачный участок”

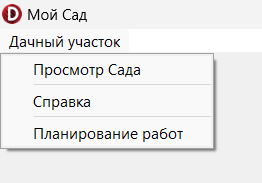


Рис. 6.9 – Разделы меню “Дачный участок”

1. В программе при выборе подменю “Справа” отобразиться небольшая справка

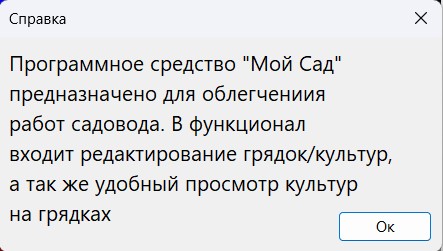


Рис. 6.10 – Справка о программном средстве.

1. Функционал программы включает в себя также просмотр садового участка

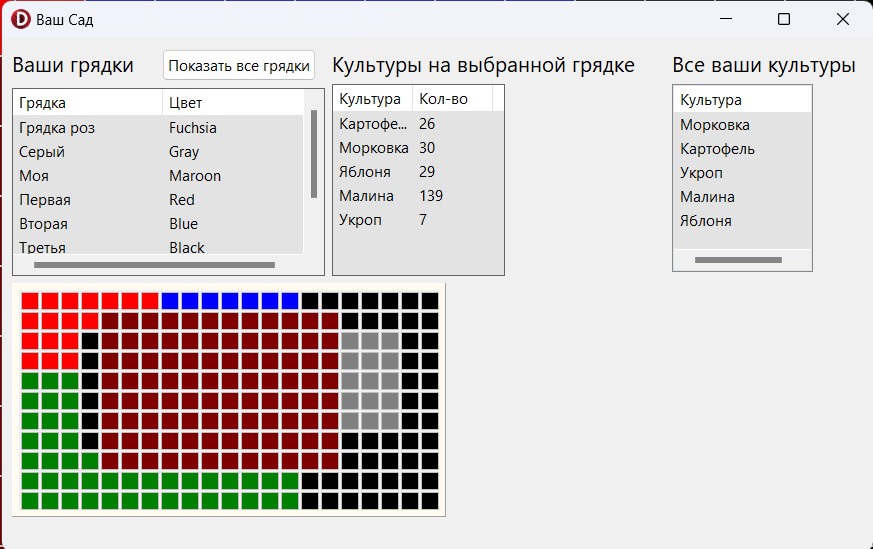


Рис. 6.11 – Просмотр дачного хозяйства.

1. Просмотр конкретной грядки

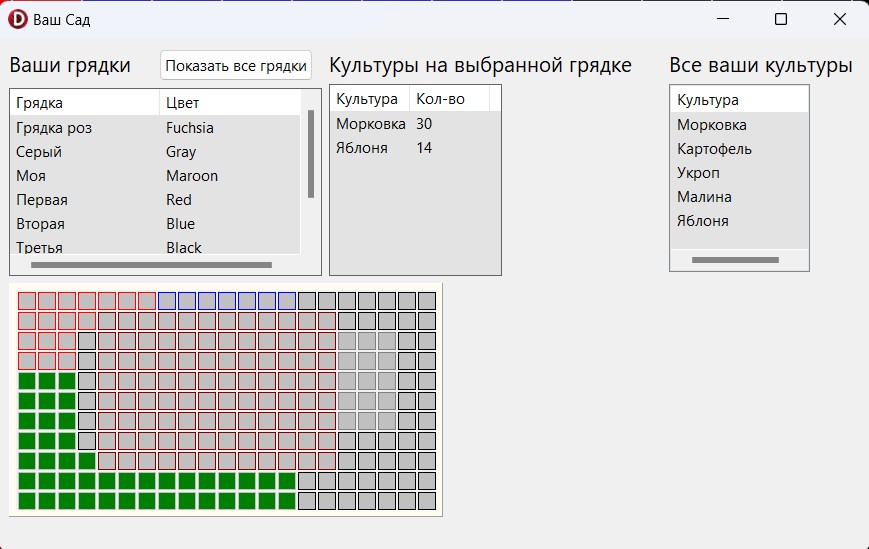


Рис. 6.12 – Отображение выбранной грядки.

1. Просмотр конкретной культуры

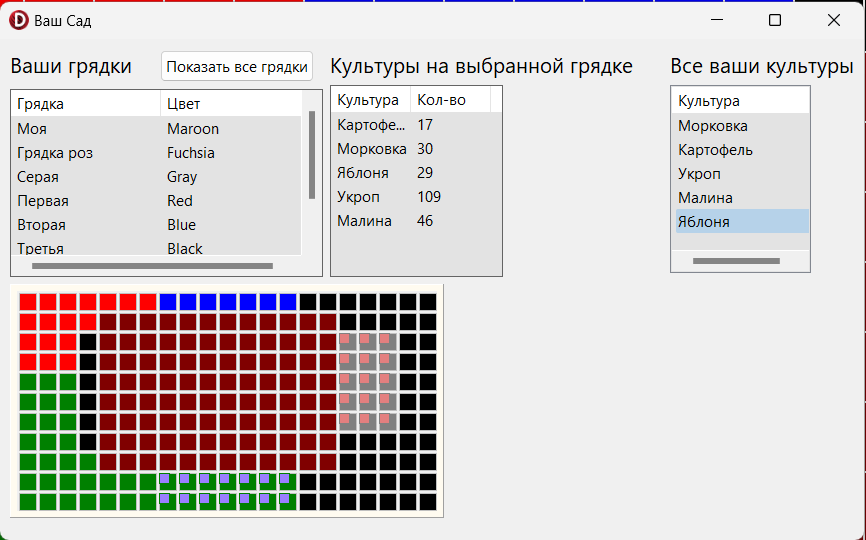


Рис. 6.13 – Отображение выбранной культуры.

1. Открытие формы добавления, просмотра или удаления работ

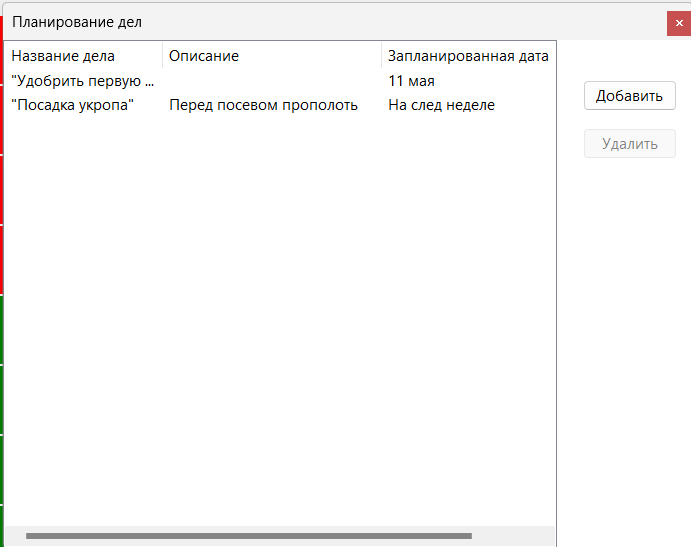


Рис. 6.14– Открытие формы просмотра и редактирования запанированных работ.

1. Добавления новой работы

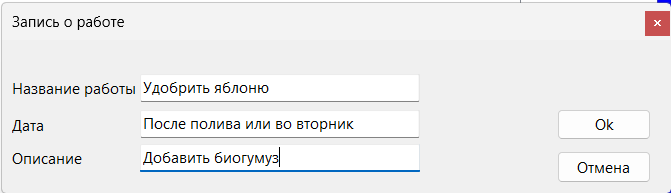


Рис. 6.15 – Открытие формы добавления работы.

1. При необходимости выйти из программы нужно нажать крестик

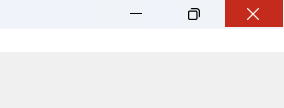


Рис. 6.16 – Закрытие программы.

**ЗАКЛЮЧЕНИЕ**

В рамках данной курсовой работы было разработано программное средство для учёта работ садовода. В ходе выполнения проекта были решены следующие основные задачи:

1. Проведён анализ предметной области и выявлены ключевые требования к функционалу программного средства учёта работ садовода.
2. Разработана структура данных для хранения информации о садовых культурах и грядках
3. Разработанное программное средство позволяет удобно и эффективно взаимодействовать с садовым участком.
4. Проведено тестирование разработанного программного средства, что позволило убедиться в корректности его работы.

Разработанное программное средство обладает высокой практической ценностью для садоводов-любителей и профессионалов. Его использование позволяет повысить производительность труда, оптимизировать расходы и улучшить качество ведения садового хозяйства. Кроме того, данное решение может быть в дальнейшем расширено дополнительными модулями, такими как интеграция с метеоданными, прогнозирование урожайности, управление теплицами и т.п.

**ПРИЛОЖЕНИЕ А**

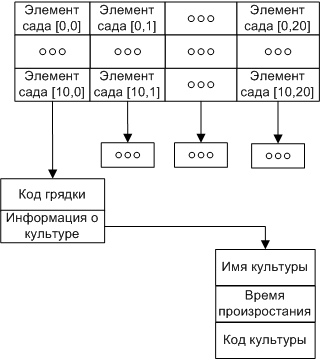


Рис. А1 – Представление данных в массиве сада

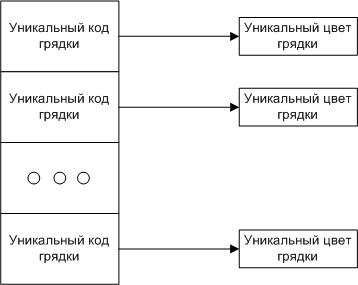


Рис. А2 – Представление данных структуре данных для хранения цвета каждой грядки

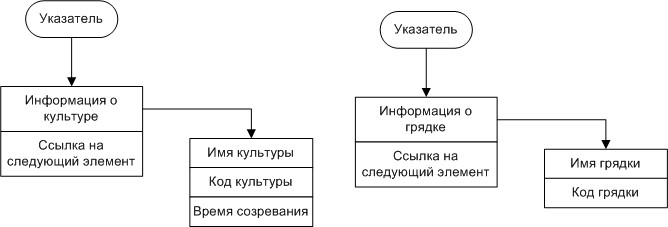


Рис. А3 – Представление данных в структурах данных всех грядок и культур

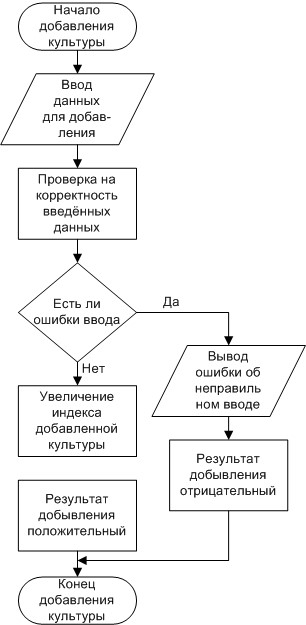


Рис. А4 – Алгоритм добавления новой культуры



Рис. А5 – Алгоритм отрисовка садового участка



Рис. А6 – Алгоритм подпрограммы по отрисовки квадрата(элемента сада и др)

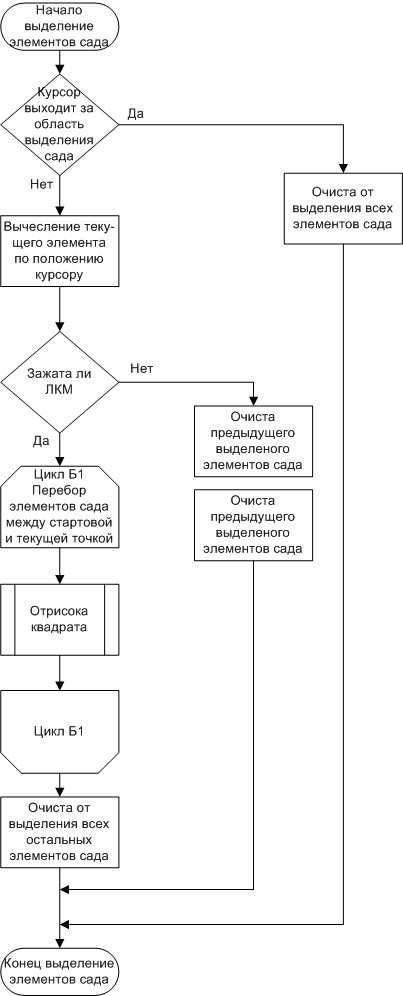


Рис. А7 – Алгоритм выделения элементов сада для изменения

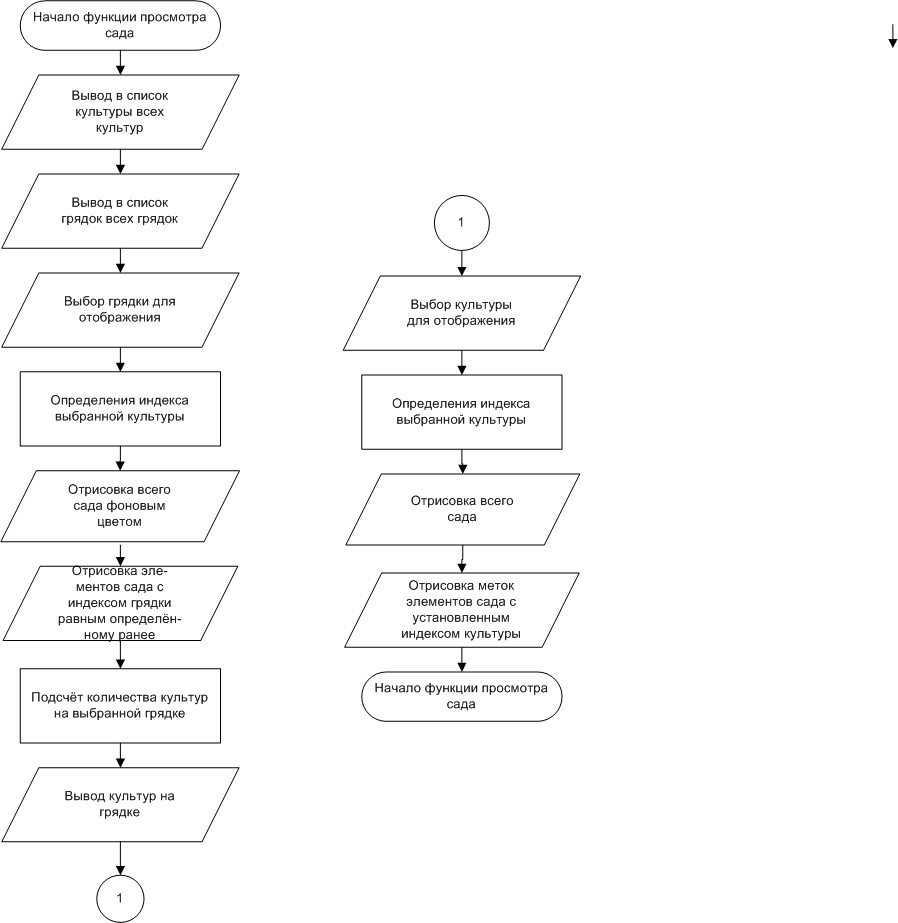


Рис. А8 – Алгоритм просмотра всего садового участка

**ПРИЛОЖЕНИЕ В**

unit uMain;

interface

uses

uGarden,

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

System.Classes, Vcl.Graphics, MMSystem, MPlayer,

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

Math, uGlobalData, uConstData, uMain\_Logical,

System.ImageList, Vcl.ImgList, System.Actions, Vcl.ActnList, Vcl.Grids,

Vcl.Menus, Vcl.ToolWin, Vcl.ComCtrls, uAddNewCulture, uAddNewGarden,

uShowALLGardens, uDictionary, Messaging, uSP,uWorks;

type

TForm1 = class(TForm)

PaintBox1: TPaintBox;

MainMenu1: TMainMenu;

N1: TMenuItem;

ToolBar1: TToolBar;

ImageList1: TImageList;

ToolButton1: TToolButton;

ActionList: TActionList;

ActionSave: TAction;

ToolButton2: TToolButton;

ToolButton3: TToolButton;

ToolButton4: TToolButton;

ActionPlus: TAction;

ActionMinus: TAction;

N8: TMenuItem;

ActionShowAll: TAction;

Label1: TLabel;

ToolButton5: TToolButton;

ToolButton6: TToolButton;

ToolButton7: TToolButton;

ActionShowSP: TAction;

N2: TMenuItem;

ShowSP1: TMenuItem;

N3: TMenuItem;

N4: TMenuItem;

ActionShowWork: TAction;

procedure PaintRect(const x, y: integer; const color: Tcolor;

const brush: Tcolor);

procedure PaintBox1Paint(Sender: TObject);

procedure ActionMinusExecute(Sender: TObject);

procedure ActionPlusExecute(Sender: TObject);

procedure PaintBox1MouseMove(Sender: TObject; Shift: TShiftState;

x, y: integer);

procedure PaintBox1Click(Sender: TObject);

procedure PaintSelection(const Point: TPoint);

procedure PaintSelectionCtrl(const Point: TPoint);

procedure PaintBox1MouseDown(Sender: TObject; Button: TMouseButton;

Shift: TShiftState; x, y: integer);

procedure PaintBox1MouseLeave(Sender: TObject);

procedure ActionSaveExecute(Sender: TObject);

procedure FormCreate(Sender: TObject);

procedure ActionShowAllExecute(Sender: TObject);

procedure FormShow(Sender: TObject);

procedure ActionShowSPExecute(Sender: TObject);

procedure ActionShowWorkExecute(Sender: TObject);

private

gardenlist: ptgarden;

culturelist: ptculture;

workslist : ptworks;

IsLBM: boolean;

ScaleGarden: integer;

CurrPoint: TPoint;

StartPoint: TPoint;

StartPointPrintGarde: TPoint;

public

end;

var

Form1: TForm1;

implementation

{$R \*.dfm}

procedure TForm1.PaintRect(const x, y: integer; const color: Tcolor;

const brush: Tcolor);

var

sizeRect: integer;

Rect: TRect;

begin

sizeRect := ScaleGarden;

PaintBox1.Canvas.Pen.color := color;

PaintBox1.Canvas.brush.color := color;

Rect.CREATE(StartPointPrintGarde.x + x \* sizeRect + PenSize,

StartPointPrintGarde.y + y \* sizeRect + PenSize, StartPointPrintGarde.x +

(x + 1) \* sizeRect, StartPointPrintGarde.y + (y + 1) \* sizeRect);

PaintBox1.Canvas.Rectangle(Rect);

end;

procedure TForm1.PaintSelection(const Point: TPoint);

var

sizeRect: integer;

Index\_x1, Index\_y1: integer;

Rect: TRect;

begin

sizeRect := ScaleGarden;

Index\_x1 := Point.x;

Index\_y1 := Point.y;

if (Index\_y1 >= 0) and (Index\_x1 >= 0) then

if (Index\_y1 <= \_GardenY) and (Index\_x1 <= \_GardenX) then

begin

PaintRect(Index\_x1, Index\_y1,

IdentifyColor(GardenMas[Index\_x1][Index\_y1].CodGarden), clSilver);

PaintBox1.Canvas.Pen.color :=

NegativeColor(IdentifyColor(GardenMas[Index\_x1][Index\_y1].CodGarden));

PaintBox1.Canvas.brush.color :=

NegativeColor(IdentifyColor(GardenMas[Index\_x1][Index\_y1].CodGarden));

Rect.CREATE(StartPointPrintGarde.x + Index\_x1 \* sizeRect + PenSize +

sizeRect div 4, StartPointPrintGarde.y + Index\_y1 \* sizeRect + PenSize +

sizeRect div 4, StartPointPrintGarde.x + (Index\_x1 + 1) \* sizeRect -

sizeRect div 4, StartPointPrintGarde.y + (Index\_y1 + 1) \* sizeRect -

sizeRect div 4);

PaintBox1.Canvas.Rectangle(Rect);

end;

end;

procedure TForm1.PaintSelectionCtrl(const Point: TPoint);

procedure ClearPole(const IndexX, IndexY: integer);

var

i, j: integer;

begin

if IndexX <= \_GardenX then

for j := StartPoint.x to IndexX do

for i := IndexY + 1 to \_GardenY do

begin

PaintRect(j, i, (IdentifyColor(GardenMas[j][i].CodGarden)), clSilver);

end;

if IndexY <= \_GardenY then

for j := StartPoint.y to \_GardenY do

for i := IndexX + 1 to \_GardenX do

begin

PaintRect(i, j, IdentifyColor(GardenMas[i][j].CodGarden), clSilver);

end;

end;

var

Index\_x1, Index\_y1: integer;

i, j: integer;

Rect: TRect;

sizeRect: integer;

begin

Index\_x1 := Point.x;

Index\_y1 := Point.y;

if (Index\_y1 >= 0) and (Index\_x1 >= 0) then

if (Index\_y1 <= \_GardenY) and (Index\_x1 <= \_GardenX) then

begin

for j := StartPoint.y to Index\_y1 do

for i := StartPoint.x to Index\_x1 do

begin

// PaintRect(i, j, NegativeColor(IdentifyColor(GardenMas[i][j].CodGarden)

// ), clWhite);

sizeRect := ScaleGarden;

PaintBox1.Canvas.Pen.color :=

NegativeColor(IdentifyColor(GardenMas[i][j].CodGarden));

PaintBox1.Canvas.brush.color :=

NegativeColor(IdentifyColor(GardenMas[i][j].CodGarden));

Rect.CREATE(StartPointPrintGarde.x + i \* sizeRect + PenSize +

sizeRect div 4, StartPointPrintGarde.y + j \* sizeRect + PenSize +

sizeRect div 4, StartPointPrintGarde.x + (i + 1) \* sizeRect -

sizeRect div 4, StartPointPrintGarde.y + (j + 1) \* sizeRect -

sizeRect div 4);

PaintBox1.Canvas.Rectangle(Rect);

end;

ClearPole(Index\_x1, Index\_y1);

end;

end;

procedure TForm1.ActionSaveExecute(Sender: TObject);

begin

SaveGardenMas;

SaveFileColor;

saveFileGarden(gardenlist);

saveFileCulture(culturelist);

SaveFileworks(workslist);;

MessageBox(Application.Handle, 'Сохранено успешно', 'Сохранение', MB\_OK);

end;

procedure TForm1.ActionShowAllExecute(Sender: TObject);

begin

Form5.FormShowAll(culturelist, gardenlist);

end;

procedure TForm1.ActionShowSPExecute(Sender: TObject);

begin

uSP.Form6.Show;

end;

procedure TForm1.ActionShowWorkExecute(Sender: TObject);

begin

Form7.MyShow(workslist);

end;

procedure TForm1.ActionMinusExecute(Sender: TObject);

begin

if ScaleGarden > 15 then

begin

dec(ScaleGarden);

Form1.PaintBox1.Invalidate;

Form1.ActionPlus.Enabled := True;

// Button\_Plus.Enabled := true;

end

else

begin

Form1.ActionMinus.Enabled := True;

// Button\_Minus.Enabled := false;

end;

end;

procedure TForm1.ActionPlusExecute(Sender: TObject);

begin

if ScaleGarden < 70 then

begin

Form1.ActionMinus.Enabled := True;

// Button\_Minus.Enabled := true;

inc(ScaleGarden);

Form1.PaintBox1.Invalidate;

end

else

begin

Form1.ActionPlus.Enabled := True;

// Button\_Plus.Enabled := false;

end;

end;

procedure TForm1.FormCreate(Sender: TObject);

var

strTemp: string;

begin

new(gardenlist);

new(culturelist);

new(workslist);

Form1.Caption := NameMainForm;

IsLBM := false;

CurrPoint.x := 0;

CurrPoint.y := 0;

StartPoint.x := -1;

StartPoint.y := -1;

StartPointPrintGarde.x := 0;

StartPointPrintGarde.y := 0;

PaintBox1.Canvas.CREATE;

ScaleGarden := 70;

PaintBox1.Width := (1 + \_GardenX) \* (ScaleGarden + PenSize) + 10;

PaintBox1.Height := (1 + \_GardenY) \* (ScaleGarden + PenSize) +10;

dictionaryColorToId := TMyDictionary<integer, Tcolor>.CREATE;

if 1 = 11 then

begin

\_CreateFirstFileGarden;

\_CreateFirstFileCulture;

\_CreateFirstFileColor;

readfileculture(culturelist);

\_CreateFirstFileGardenMas(culturelist);

end;

try

ReadFileColor;

ReadfileWorks(workslist);

readGardenMas;

readfileculture(culturelist);

readfilegarden(gardenlist);

except

ShowMessage('ex : readfile');

end;

end;

procedure TForm1.FormShow(Sender: TObject);

begin

//

end;

procedure TForm1.PaintBox1MouseLeave(Sender: TObject);

begin

if (CurrPoint.y <= \_GardenY) and (CurrPoint.x <= \_GardenX) then

if (CurrPoint.y >= 0) and (CurrPoint.x >= 0) then

if not IsLBM then

begin

PaintRect(CurrPoint.x, CurrPoint.y,

IdentifyColor(GardenMas[CurrPoint.x][CurrPoint.y].CodGarden),

clSilver);

end;

end;

procedure TForm1.PaintBox1Click(Sender: TObject);

begin

if (StartPoint.x <> -1) and (StartPoint.y <> -1) then

begin

form2.createLists(culturelist, gardenlist);

if (abs(StartPoint.x - CurrPoint.x) <> 0) or

(abs(StartPoint.y - CurrPoint.y) <> 0) then

begin

if (StartPoint.x <= CurrPoint.x) and (StartPoint.y <= CurrPoint.y) then

uGarden.form2.FormShowForChangeCtrl(CurrPoint, StartPoint);

end

else

begin

uGarden.form2.FormShowForChange(CurrPoint);

end;

IsLBM := false;

Invalidate;

StartPoint.x := -1;

StartPoint.y := -1;

CurrPoint.x := 0;

CurrPoint.y := 0;

end;

end;

procedure TForm1.PaintBox1MouseDown(Sender: TObject; Button: TMouseButton;

Shift: TShiftState; x, y: integer);

begin

// StartPoint.x := -1;

// StartPoint.y := -1;

if (StartPointPrintGarde.y < y) and (StartPointPrintGarde.x < x) then

if ((1 + \_GardenX) \* (ScaleGarden + PenSize) >= x) and

((1 + \_GardenY) \* (ScaleGarden + PenSize) >= y) then

if (Button = mbleft) then

begin

IsLBM := True;

if (StartPoint.y = -1) or (StartPoint.x = -1) then

begin

StartPoint.x := (x - StartPointPrintGarde.x) div ScaleGarden;

StartPoint.y := (y - StartPointPrintGarde.y) div ScaleGarden;

end;

end;

end;

procedure TForm1.PaintBox1MouseMove(Sender: TObject; Shift: TShiftState;

x, y: integer);

var

newPoint: TPoint;

begin

if (StartPointPrintGarde.y < y) and (StartPointPrintGarde.x < x) then

begin

if ((1 + \_GardenX) \* (ScaleGarden + PenSize) >= x) and

((1 + \_GardenY) \* (ScaleGarden + PenSize) >= y) then

begin

if (IsLBM) then

begin

CurrPoint.x := (x - StartPointPrintGarde.x) div ScaleGarden;

CurrPoint.y := (y - StartPointPrintGarde.y) div ScaleGarden;

if (CurrPoint.y <= \_GardenY) and (CurrPoint.x <= \_GardenX) then

if (CurrPoint.y >= 0) and (CurrPoint.x >= 0) then

begin

PaintSelectionCtrl(CurrPoint);

end;

end

else

begin

newPoint.x := (x - StartPointPrintGarde.x) div (ScaleGarden);

newPoint.y := (y - StartPointPrintGarde.y) div (ScaleGarden);

if (newPoint.x <> CurrPoint.x) or (newPoint.y <> CurrPoint.y) then

begin

if (CurrPoint.y <= \_GardenY) and (CurrPoint.x <= \_GardenX) then

if (CurrPoint.y >= 0) and (CurrPoint.x >= 0) then

begin

PaintRect(CurrPoint.x, CurrPoint.y,

IdentifyColor(GardenMas[CurrPoint.x][CurrPoint.y].CodGarden),

clSilver);

end;

CurrPoint := newPoint;

if (CurrPoint.y <= \_GardenY) and (CurrPoint.x <= \_GardenX) then

if (CurrPoint.y >= 0) and (CurrPoint.x >= 0) then

begin

PaintSelection(CurrPoint);

end;

end;

end;

end

else

begin

if (CurrPoint.y <= \_GardenY) and (CurrPoint.x <= \_GardenX) then

if (CurrPoint.y >= 0) and (CurrPoint.x >= 0) then

if not IsLBM then

begin

PaintRect(CurrPoint.x, CurrPoint.y,

IdentifyColor(GardenMas[CurrPoint.x][CurrPoint.y].CodGarden),

clSilver);

end;

CurrPoint.x := -1;

CurrPoint.y := -1;

end;

end

else

begin

if (CurrPoint.y <= \_GardenY) and (CurrPoint.x <= \_GardenX) then

if (CurrPoint.y >= 0) and (CurrPoint.x >= 0) then

if not IsLBM then

begin

PaintRect(CurrPoint.x, CurrPoint.y,

IdentifyColor(GardenMas[CurrPoint.x][CurrPoint.y].CodGarden),

clSilver);

end;

CurrPoint.x := -1;

CurrPoint.y := -1;

end;

end;

procedure TForm1.PaintBox1Paint(Sender: TObject);

begin

PaintBox1.Canvas.brush.color := clBtnFace;

PaintBox1.Canvas.FillRect(PaintBox1.ClientRect);

StartPointPrintGarde.x :=

(PaintBox1.Width - \_GardenX \* (ScaleGarden + PenSize)) div 2 + 5;

StartPointPrintGarde.y :=

(PaintBox1.Height - \_GardenY \* (ScaleGarden + PenSize)) div 2 - 5;

for var i := 0 to \_GardenX do

begin

for var j := 0 to \_GardenY do

begin

PaintRect(i, j, IdentifyColor(GardenMas[i][j].CodGarden), clSilver);

end;

end;

end;

end.

**unit uGarden;**

**interface**

**uses**

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

**System.Classes, Vcl.Graphics,**

**Vcl.Controls, Vcl.Forms, Vcl.Dialogs, uGlobalData, Vcl.StdCtrls, uConstData,**

**uGarden\_Logical, Vcl.Menus, Vcl.Grids, uAddNewCulture, uAddNewGarden,**

**Vcl.ComCtrls,**

**System.Actions, Vcl.ActnList, uDictionary;**

**type**

**TForm2 = class(TForm)**

**LabelCulture: TLabel;**

**LabelTime: TLabel;**

**LabelBed: TLabel;**

**ButtonSave: TButton;**

**EditCulture: TEdit;**

**EditTime: TEdit;**

**EditGarden: TEdit;**

**LabelGarden: TLabel;**

**Label2: TLabel;**

**ButtonClose: TButton;**

**ButtonAddCulture: TButton;**

**ButtonAddGarden: TButton;**

**ListViewCulture: TListView;**

**ListViewGarden: TListView;**

**ButtonDeleteGarden: TButton;**

**ButtonDeleteCulture: TButton;**

**Button1: TButton;**

**ActionListForm2: TActionList;**

**ActionAddCulture: TAction;**

**ActionAddGarden: TAction;**

**ActionDeleteCulture: TAction;**

**ActionDeleteGarden: TAction;**

**procedure ButtonSaveClick(Sender: TObject);**

**procedure ButtonCloseClick(Sender: TObject);**

**procedure ListViewCultureClick(Sender: TObject);**

**procedure CreateListViewGarden(list: PtGarden);**

**procedure CreateListViewCulture(list: PtCulture);**

**procedure ListViewGardenClick(Sender: TObject);**

**procedure Button1Click(Sender: TObject);**

**procedure ActionAddCultureExecute(Sender: TObject);**

**procedure ActionAddGardenExecute(Sender: TObject);**

**procedure ActionDeleteCultureExecute(Sender: TObject);**

**procedure ActionDeleteGardenExecute(Sender: TObject);**

**private**

**CurrPoint: TPoint;**

**StartPoint: TPoint;**

**culturelist: PtCulture;**

**gardenlist: PtGarden;**

**{ Private declarations }**

**public**

**{ Public declarations }**

**function FormShowForChange(CurrPoint: TPoint): TModalResult;**

**function FormShowForChangeCtrl(CurrPoint, StartPoint: TPoint): TModalResult;**

**procedure createLists(culturelist: PtCulture; gardenlist: PtGarden);**

**end;**

**var**

**Form2: TForm2;**

**implementation**

**{$R \*.dfm}**

**procedure TForm2.createLists(culturelist: PtCulture; gardenlist: PtGarden);**

**begin**

**Form2.culturelist := culturelist;**

**Form2.gardenlist := gardenlist;**

**end;**

**procedure TForm2.CreateListViewGarden(list: PtGarden);**

**var**

**ListItem: TListItem;**

**strColor: string;**

**begin**

**ListViewGarden.Clear;**

**ListViewGarden.Columns[0].Width := 100;**

**ListViewGarden.Columns[1].Width := 100;**

**list := list.Next;**

**while list <> nil do**

**begin**

**ListItem := ListViewGarden.Items.Add;**

**ListItem.Caption := list.garden.Name;**

**strColor := ColorToString(IdentifyColor(list.garden.CodGarden));**

**ListItem.SubItems.Add(Copy(strColor, 3, Length(strColor)));**

**list := list.Next;**

**end;**

**end;**

**procedure TForm2.CreateListViewCulture(list: PtCulture);**

**var**

**ListItem: TListItem;**

**begin**

**ListViewCulture.Clear;**

**ListViewCulture.Columns[0].Width := 100;**

**ListViewCulture.Columns[1].Width := 200;**

**list := list.Next;**

**while list <> nil do**

**begin**

**ListItem := ListViewCulture.Items.Add;**

**// ShowMessage(list.culture.Name);**

**ListItem.Caption := list.culture.Name;**

**ListItem.SubItems.Add(ConvertDateToString(list.culture.Time));**

**list := list.Next;**

**end;**

**end;**

**procedure TForm2.ActionAddCultureExecute(Sender: TObject);**

**var**

**Res: TModalResult;**

**newculture: cultureListInfo;**

**begin**

**Res := Form3.ShowForAdd(newculture, culturelist);**

**if Res = mrnone then**

**Exit;**

**// ShowMessage('add culture result : ' + inttostr(ord(Res)));**

**AddCulture(newculture, culturelist);**

**CreateListViewCulture(culturelist);**

**end;**

**procedure TForm2.ActionAddGardenExecute(Sender: TObject);**

**var**

**Res: TModalResult;**

**newgarden: TGARDEN;**

**begin**

**Res := Form4.ShowForAdd(newgarden, gardenlist);**

**if Res = mrnone then**

**Exit;**

**// ShowMessage('add garden');**

**AddGarden(newgarden, gardenlist);**

**CreateListViewGarden(gardenlist);**

**end;**

**procedure TForm2.ActionDeleteCultureExecute(Sender: TObject);**

**var**

**SelectedItem: TListItem;**

**begin**

**if ListViewCulture.Selected <> nil then**

**begin**

**SelectedItem := ListViewCulture.Selected;**

**if Assigned(SelectedItem) then**

**begin**

**if CheckCultureUsing(SelectedItem.Caption) then**

**begin**

**MessageBox(Application.Handle,**

**'Данная культура уже используется в хозяйтсве',**

**'Ошибка удаления', MB\_OK);**

**end**

**else**

**begin**

**DeleteCulture(SelectedItem.Caption, culturelist);**

**CreateListViewCulture(culturelist);**

**end;**

**end;**

**end**

**else**

**begin**

**ShowMessage('Ex');**

**end;**

**ButtonDeleteCulture.Enabled := ListViewCulture.Selected <> nil;**

**end;**

**procedure TForm2.ActionDeleteGardenExecute(Sender: TObject);**

**var**

**SelectedItem: TListItem;**

**begin**

**if ListViewGarden.Selected <> nil then**

**begin**

**SelectedItem := ListViewGarden.Selected;**

**if Assigned(SelectedItem) then**

**begin**

**if CheckGardenUsing(getidgarden(SelectedItem.Caption, gardenlist)) then**

**begin**

**MessageBox(Application.Handle,**

**'Данная грядка уже используется в хозяйтсве',**

**'Ошибка удаления', MB\_OK);**

**end**

**else**

**begin**

**DeleteGarden(SelectedItem.Caption, gardenlist);**

**CreateListViewGarden(gardenlist);**

**end;**

**end;**

**end**

**else**

**begin**

**ShowMessage('Ex');**

**end;**

**ButtonDeleteGarden.Enabled := ListViewCulture.Selected <> nil;**

**end;**

**procedure TForm2.Button1Click(Sender: TObject);**

**begin**

**// ShowMessage('currMaxIdGarden : ' + inttostr(currMaxIdGarden));**

**PrintCulture(culturelist);**

**PrintGarden(dictionaryColorToId, gardenlist);**

**PrintDictionary;**

**end;**

**procedure TForm2.ButtonCloseClick(Sender: TObject);**

**begin**

**Close();**

**end;**

**procedure TForm2.ButtonSaveClick(Sender: TObject);**

**var**

**culture: TGardenCell;**

**strNameGarden: string[50];**

**strNameCulture: string[50];**

**I: integer;**

**strMessage: string;**

**begin**

**strMessage := '';**

**strNameCulture := EditCulture.Text;**

**strNameGarden := EditGarden.Text;**

**if not ConvertStringToDate(EditTime.Text, culture.Сulture.Time) then**

**strMessage := 'Неверная дата';**

**if not IsValidGarden(gardenlist, strNameGarden, culture.CodGarden) then**

**strMessage := strMessage + #13#10 + 'Неверное имя грядки';**

**if not IsValidCulture(culturelist, strNameCulture, culture.Сulture.cod) then**

**strMessage := strMessage + #13#10 + 'Неверное имя культуры';**

**culture.Сulture.Name := strNameCulture;**

**if (strMessage = '') then**

**begin**

**for var K := StartPoint.X to CurrPoint.X do**

**for var J := StartPoint.Y to CurrPoint.Y do**

**begin**

**gardenmas[K][J] := culture;**

**end;**

**Close();**

**end**

**else**

**begin**

**MessageBox(Application.Handle, PChar(strMessage), 'Ошибка ввода', MB\_OK);**

**end;**

**end;**

**function TForm2.FormShowForChange(CurrPoint: TPoint): TModalResult;**

**function IdentifyGardenName(list: PtGarden; const cod: integer): string;**

**begin**

**list := list.Next;**

**while list <> nil do**

**begin**

**if list.garden.CodGarden = cod then**

**begin**

**result := list.garden.Name;**

**break;**

**end;**

**list := list.Next;**

**end;**

**end;**

**function IdentifyCultureName(list: PtCulture; const cod: integer): string;**

**begin**

**list := list.Next;**

**while list <> nil do**

**begin**

**if list.culture.cod = cod then**

**begin**

**result := list.culture.Name;**

**break;**

**end;**

**list := list.Next;**

**end;**

**end;**

**begin**

**CreateListViewGarden(gardenlist);**

**CreateListViewCulture(culturelist);**

**Form2.Caption := 'Изменение грядки';**

**ButtonDeleteCulture.Enabled := ListViewCulture.Selected <> nil;**

**ButtonDeleteGarden.Enabled := ListViewGarden.Selected <> nil;**

**Form2.CurrPoint := CurrPoint;**

**Form2.StartPoint := CurrPoint;**

**EditCulture.Text := IdentifyCultureName(culturelist,**

**gardenmas[CurrPoint.X][CurrPoint.Y].Сulture.cod);**

**EditTime.Text := ConvertDateToString(gardenmas[CurrPoint.X][CurrPoint.Y]**

**.Сulture.Time);**

**EditGarden.Text := IdentifyGardenName(gardenlist,**

**gardenmas[CurrPoint.X][CurrPoint.Y].CodGarden);**

**result := ShowModal;**

**end;**

**function TForm2.FormShowForChangeCtrl(CurrPoint, StartPoint: TPoint): TModalResult;**

**begin**

**CreateListViewGarden(gardenlist);**

**CreateListViewCulture(culturelist);**

**Form2.Caption := 'Изменение грядок';**

**Form2.CurrPoint := CurrPoint;**

**Form2.StartPoint := StartPoint;**

**EditCulture.Text := '';**

**EditTime.Text := '';**

**EditGarden.Text := '';**

**result := ShowModal;**

**end;**

**procedure TForm2.ListViewCultureClick(Sender: TObject);**

**var**

**SelectedItem: TListItem;**

**begin**

**ButtonDeleteCulture.Enabled := ListViewCulture.Selected <> nil;**

**SelectedItem := ListViewCulture.Selected;**

**if Assigned(SelectedItem) then**

**begin**

**EditCulture.Text := SelectedItem.Caption;**

**EditTime.Text := SelectedItem.SubItems[0];**

**end;**

**end;**

**procedure TForm2.ListViewGardenClick(Sender: TObject);**

**var**

**SelectedItem: TListItem;**

**begin**

**ButtonDeleteGarden.Enabled := ListViewGarden.Selected <> nil;**

**SelectedItem := ListViewGarden.Selected;**

**if Assigned(SelectedItem) then**

**begin**

**EditGarden.Text := SelectedItem.Caption;**

**end;**

**end;**

**end.**

unit uGlobalData;

interface

uses

Types, Graphics, sysUtils,

Vcl.Dialogs, uDictionary,windows;

const

textSP = 'Программное средство "Мой Сад"' + #13#10 +

'предназначено для облегчениия' + #13#10 + 'работ садовода. В функционал ' +

#13#10 + 'входит редактирование грядок/культур,' + #13#10 +

'а так же удобный просмотр культур' + #13#10 + 'на грядках';

PenSize = 2;

FileCulture = 'culture.cul';

FileGarden = 'garden.cul';

FileColor = 'colorsGarden.cul';

NameFileWorks = 'worksDeal.cul';

UserFileGarden = 'userGarden.garden';

NameMainForm = 'Мой Сад';

\_GardenX = 20;

\_GardenY = 10;

type

worksRecord = record

Name: string[255];

Description: string[255];

Date: string[255];

end;

ptworks = ^works;

works = record

work: worksRecord;

next: ptworks;

end;

Tstring = string[50];

TMyDate = record

Year: integer;

Month: integer;

Days: integer;

end;

Ptculture = ^cultureRecord;

cultureListInfo = record

Name: string[50];

Time: TMyDate;

cod: integer;

end;

cultureRecord = record

culture: cultureListInfo;

Next: Ptculture;

end;

PtGarden = ^GardenRecord;

TGarden = record

CodGarden: integer;

Name: string[50];

end;

GardenRecord = record

garden: TGarden;

Next: PtGarden;

end;

TGardenCell = record

CodGarden: integer;

Сulture: cultureListInfo;

end;

RecordDictionary = record

Color: TColor;

corGarden: integer;

end;

TgardenMas = array [0 .. \_GardenX] of array [0 .. \_GardenY] of TGardenCell;

var

dictionaryColorToId: TMyDictionary<integer, TColor>;

GardenMas: TgardenMas;

currMaxIdCulture: integer;

currMaxIdGarden: integer;

function IdentifyColor(const cod: integer): TColor;

function ConvertStringToDate(const dateStr: string; var Date: TMyDate): boolean;

function ConvertDateToString(const dateStr: TMyDate): string;

function GetIdGarden(const Name: string; list: PtGarden): integer;

procedure PrintGarden(dictionaryColorToId: TMyDictionary<integer, TColor>;

list: PtGarden);

procedure PrintDictionary;

procedure PrintCulture(list: Ptculture);

function NegativeColor(const color: Tcolor): Tcolor;

implementation

function NegativeColor(const color: Tcolor): Tcolor;

var

R, G, B: BYTE;

begin

randomize;

result := RGB((255 - GetRValue(color)) mod 256, (255 - GetGValue(color))

mod 256, (255 - GetBValue(color)) mod 256);

if GetRValue(result) > 128 then

begin

result := RGB(GetRValue(result) - 100, GetGValue(result),

GetBValue(result));

end

else

begin

result := RGB(GetRValue(result) + 100, GetGValue(result),

GetBValue(result));

end;

end ;

function GetIdGarden(const Name: string; list: PtGarden): integer;

begin

list := list.Next;

result := -1;

try

while list <> nil do

begin

if list.garden.Name = Name then

begin

result := list.garden.CodGarden;

break;

end;

list := list.Next;

end;

except

ShowMessage('Такого имени в грядках не существует');

end;

if result = -1 then

begin

ShowMessage('Такого имени в грядках не существует');

end;

end;

procedure PrintDictionary;

var

str: string;

begin

str := 'Dictionary' + #13#10;

for var temp in dictionaryColorToId.GetAllItems do

begin

str := str + #13#10 + inttostr(temp.Key) + ' | ' +

ColorToString(temp.Value);

end;

ShowMessage(str);

end;

procedure PrintGarden(dictionaryColorToId: TMyDictionary<integer, TColor>;

list: PtGarden);

var

str: string;

begin

str := 'Garden' + #13#10;

list := list.Next;

while list <> nil do

begin

str := str + #13#10 + inttostr(list.garden.CodGarden) + ' | ' +

list.garden.Name + ' --> ';

if dictionaryColorToId.ContainsKey(list.garden.CodGarden) then

begin

str := str + ColorToString(dictionaryColorToId[list.garden.CodGarden]);

end

else

begin

str := str + 'error';

end;

list := list.Next;

end;

ShowMessage(str);

end;

procedure PrintCulture(list: Ptculture);

var

str: string;

begin

str := 'culture' + #13#10;

list := list.Next;

while list <> nil do

begin

str := str + #13#10 + inttostr(list.culture.cod) + ' | ' +

list.culture.Name + ' | ' + ConvertDateToString(list.culture.Time);

list := list.Next;

end;

ShowMessage(str);

end;

function IdentifyColor(const cod: integer): TColor;

begin

result := dictionaryColorToId[cod];

end;

function ConvertStringToDate(const dateStr: string; var Date: TMyDate): boolean;

begin

ConvertStringToDate := true;

try

Date.Days := StrToInt((dateStr[1] + dateStr[2]));

Date.Month := StrToInt((dateStr[4] + dateStr[5]));

Date.Year := StrToInt((dateStr[7]));

except

ConvertStringToDate := false;

end;

if (Length(dateStr) <> 7) or (dateStr[3] <> ':') or (dateStr[6] <> ':') then

ConvertStringToDate := false;

end;

function ConvertDateToString(const dateStr: TMyDate): string;

begin

result := inttostr(dateStr.Days);

if dateStr.Days < 10 then

begin

result := '0' + result;

end;

result := result + ':';

if dateStr.Month < 10 then

begin

result := result + '0';

end;

result := result + inttostr(dateStr.Month) + ':' + inttostr(dateStr.Year);

end;

end.

unit uGarden\_Logical;

interface

uses

uConstData, uGLOBALdATA, SysUtils, Vcl.Dialogs,

Graphics, udictionary;

function CheckCultureUsing(const culture: string): boolean;

function CheckGardenUsing(const gardencod: integer): boolean;

procedure DeleteCulture(const name: string; list: Ptculture);

procedure DeleteGarden(const name: string; list: ptgarden);

procedure AddCulture(const newCulture: cultureListInfo; list: Ptculture);

procedure AddGarden(const newgARDEN: TGarden; list: ptgarden);

function IsValidGarden(list: ptgarden; const str: string;

var cod: integer): boolean;

function IsValidCulture(list: Ptculture; const str: string;

var cod: integer): boolean;

implementation

procedure AddGarden(const newgARDEN: TGarden; list: ptgarden);

var

firstNode: ptgarden;

begin

firstNode := list.Next;

new(list.Next);

list.Next.garden := newgARDEN;

list.Next.Next := firstNode;

end;

function IsValidGarden(list: ptgarden; const str: string;

var cod: integer): boolean;

begin

result := false;

list := list.Next;

while list <> nil do

begin

if list.garden.name = str then

begin

result := true;

cod := list.garden.CodGarden;

break;

end;

list := list.Next;

end;

end;

function IsValidCulture(list: Ptculture; const str: string;

var cod: integer): boolean;

begin

result := false;

list := list.Next;

while list <> nil do

begin

if list.culture.name = str then

begin

result := true;

cod := list.culture.cod;

break;

end;

list := list.Next;

end;

end;

procedure AddCulture(const newCulture: cultureListInfo; list: Ptculture);

var

firstNode: Ptculture;

begin

firstNode := list.Next;

new(list.Next);

list.Next.culture := newCulture;

list.Next.Next := firstNode;

end;

function CheckCultureUsing(const culture: string): boolean;

begin

result := false;

for var I := 0 to \_GardenX do

begin

for var J := 0 to \_GardenY do

begin

if GardenMas[I][J].Сulture.name = culture then

begin

result := true;

end;

end;

end;

end;

procedure DeleteCulture(const name: string; list: Ptculture);

begin

while (list.Next <> nil) and (list.Next.culture.name <> name) do

begin

list := list.Next;

end;

if (list.Next <> nil) and (list.Next.culture.name = name) then

begin

list.Next := list.Next.Next;

end;

end;

function CheckGardenUsing(const gardencod: integer): boolean;

begin

result := false;

for var I := 0 to \_GardenX do

begin

for var J := 0 to \_GardenY do

begin

if GardenMas[I][J].CodGarden = gardencod then

begin

result := true;

end;

end;

end;

end;

procedure DeleteGarden(const name: string; list: ptgarden);

var

index: integer;

begin

index := GetIdGarden(name, list);

// ShowMessage('Имя на удаление : ' + name + ' индекс : ' + inttostr(index));

while (list.Next <> nil) and (list.Next.garden.CodGarden <> index) do

begin

list := list.Next;

end;

if (list.Next <> nil) and (list.Next.garden.CodGarden = index) then

begin

list.Next := list.Next.Next;

// ShowMessage('id : ' + inttostr(index) + ' color : ' +

// ColorToString(dictionaryColorToId[index]));

dictionaryColorToId.Remove(index);

end;

end;

end.

unit uMain\_Logical;

interface

uses

uglobalData, uconstdata, Vcl.Graphics, uDictionary;

procedure SaveFileGarden(list: PtGarden);

procedure ReadFileGarden(list: PtGarden);

// \*

procedure ReadFileCulture(list: Ptculture);

procedure SaveFileCulture(list: Ptculture);

// \*

procedure ReadFileColor;

procedure SaveFileColor;

// \*

procedure SaveGardenMas;

procedure ReadGardenMas;

procedure SaveFileWorks(list: ptworks);

procedure ReadfileWorks(list: ptworks);

// \*

// ONLY FOR DEVELOPERS

// {

procedure \_CreateFirstFileGardenMas(culturelist: Ptculture);

procedure \_CreateFirstFileCulture;

procedure \_CreateFirstFileGarden;

procedure \_CreateFirstFileColor;

// }

implementation

procedure ReadFileCulture(list: Ptculture);

var

currFile: file of cultureListInfo;

count: integer;

begin

count := 0;

AssignFile(currFile, FileCulture);

Reset(currFile);

while not EoF(currFile) do

begin;

new(list^.Next);

list := list^.Next;

Read(currFile, list^.culture);

if count < list^.culture.cod then

begin

count := list^.culture.cod;

end;

end;

list^.Next := nil;

currMaxIdCulture := count;

Closefile(currFile);

end;

procedure ReadfileWorks(list: ptworks);

var

currFile: file of worksRecord;

begin

AssignFile(currFile, NameFileWorks);

reset(currFile);

while not EoF(currFile) do

begin;

new(list^.next);

list := list^.next;

Read(currFile, list^.work);

end;

list := nil;

Closefile(currFile);

end;

procedure SaveFileWorks(list: ptworks);

var

currFile: file of worksRecord;

count: integer;

begin

AssignFile(currFile, NameFileWorks);

Rewrite(currFile);

count := 0;

list := list.Next;

while (list <> nil) do

begin

seek(currFile, count);

write(currFile, list.work);

inc(count);

list := list.Next;

end;

Closefile(currFile);

end;

procedure SaveFileCulture(list: Ptculture);

var

currFile: file of cultureListInfo;

count: integer;

begin

AssignFile(currFile, FileCulture);

Rewrite(currFile);

count := 0;

list := list.Next;

while (list <> nil) do

begin

seek(currFile, count);

write(currFile, list.culture);

inc(count);

list := list.Next;

end;

Closefile(currFile);

end;

procedure SaveFileGarden(list: PtGarden);

var

currFile: file of TGarden;

count: integer;

begin

AssignFile(currFile, FileGarden);

Rewrite(currFile);

count := 0;

list := list.Next;

while (list <> nil) do

begin

seek(currFile, count);

write(currFile, list.garden);

inc(count);

list := list.Next;

end;

Closefile(currFile);

end;

procedure ReadFileGarden(list: PtGarden);

var

currFile: file of TGarden;

count: integer;

begin

count := 0;

AssignFile(currFile, FileGarden);

Reset(currFile);

while not EoF(currFile) do

begin

new(list^.Next);

list := list^.Next;

Read(currFile, list^.garden);

count := list^.garden.CodGarden;

end;

list^.Next := nil;

Closefile(currFile);

end;

procedure \_CreateFirstFileColor;

var

currFile: file of RecordDictionary;

countStr: integer;

tempPair: RecordDictionary;

begin

AssignFile(currFile, FileColor);

Rewrite(currFile);

countStr := 0;

seek(currFile, countStr);

tempPair.Color := clred;

tempPair.corGarden := 1;

write(currFile, tempPair);

inc(countStr);

seek(currFile, countStr);

tempPair.Color := clblue;

tempPair.corGarden := 2;

write(currFile, tempPair);

inc(countStr);

seek(currFile, countStr);

tempPair.Color := clblack;

tempPair.corGarden := 3;

write(currFile, tempPair);

inc(countStr);

seek(currFile, countStr);

tempPair.Color := clgreen;

tempPair.corGarden := 4;

write(currFile, tempPair);

inc(countStr);

Close(currFile);

end;

procedure \_CreateFirstFileCulture;

function CreateDate(const AYear, AMonth, ADays: integer): TMyDate;

begin

CreateDate.Year := AYear;

CreateDate.Month := AMonth;

CreateDate.Days := ADays;

end;

var

currFile: file of cultureListInfo;

Str: cultureListInfo;

countStr: integer;

begin

countStr := 0;

AssignFile(currFile, FileCulture);

Rewrite(currFile);

seek(currFile, countStr);

Str.name := 'Морковка';

Str.cod := 1;

Str.Time := CreateDate(1, 1, 1);

write(currFile, Str);

inc(countStr);

seek(currFile, countStr);

Str.name := 'Картофель';

Str.cod := 2;

Str.Time := CreateDate(2, 2, 2);

write(currFile, Str);

inc(countStr);

seek(currFile, countStr);

Str.name := 'Укроп';

Str.cod := 3;

Str.Time := CreateDate(3, 3, 3);

write(currFile, Str);

inc(countStr);

seek(currFile, countStr);

Str.name := 'Малина';

Str.cod := 4;

Str.Time := CreateDate(4, 4, 4);

write(currFile, Str);

inc(countStr);

seek(currFile, countStr);

Str.name := 'Яблоня';

Str.cod := 5;

Str.Time := CreateDate(5, 5, 5);

write(currFile, Str);

inc(countStr);

Closefile(currFile);

end;

procedure \_CreateFirstFileGarden;

var

currFile: file of TGarden;

Str: TGarden;

countStr: integer;

begin

countStr := 0;

AssignFile(currFile, FileGarden);

Rewrite(currFile);

seek(currFile, countStr);

Str.name := 'Первая';

Str.CodGarden := 1;

write(currFile, Str);

inc(countStr);

seek(currFile, countStr);

Str.name := 'Вторая';

Str.CodGarden := 2;

write(currFile, Str);

inc(countStr);

seek(currFile, countStr);

Str.name := 'Третья';

Str.CodGarden := 3;

write(currFile, Str);

inc(countStr);

seek(currFile, countStr);

Str.name := 'Четвёртая';

Str.CodGarden := 4;

write(currFile, Str);

inc(countStr);

Closefile(currFile);

end;

procedure SaveFileColor;

var

currFile: file of RecordDictionary;

count: integer;

tempPair: RecordDictionary;

begin

count := 0;

AssignFile(currFile, FileColor);

Rewrite(currFile);

for var temp in dictionaryColorToId.GetAllItems do

begin

tempPair.corGarden := temp.Key;

tempPair.Color := temp.Value;

seek(currFile, count);

write(currFile, tempPair);

inc(count);

end;

Closefile(currFile);

end;

procedure SaveGardenMas;

var

currFile: file of TGardenCell;

count: integer;

begin

AssignFile(currFile, UserFileGarden);

Rewrite(currFile);

count := 0;

for var I := 0 to \_GardenX do

begin

for var J := 0 to \_GardenY do

begin

seek(currFile, count);

write(currFile, GardenMas[I][J]);

inc(count);

end;

end;

Closefile(currFile);

end;

procedure \_CreateFirstFileGardenMas(culturelist: Ptculture);

var

GardenMas: TgardenMas;

I, J: integer;

currFile: file of TGardenCell;

count: integer;

Str: TGardenCell;

list: Ptculture;

begin

// \*

for I := 0 to 6 do

begin

for J := 0 to 4 do

begin

GardenMas[I][J].CodGarden := 1;

GardenMas[I][J].Сulture.cod := 2;

// GardenMas[I][J].Сulture.Name := 666;

end;

end;

for I := 7 to 13 do

begin

for J := 0 to 7 do

begin

GardenMas[I][J].CodGarden := 2;

GardenMas[I][J].Сulture.cod := 3;

// GardenMas[I][J].Сulture := 666;

end;

end;

for I := 14 to \_GardenX do

begin

for J := 0 to \_GardenY do

begin

GardenMas[I][J].CodGarden := 3;

GardenMas[I][J].Сulture.cod := 4;

// GardenMas[I][J].Сulture := 666;

end;

end;

for I := 7 to 13 do

begin

for J := 6 to \_GardenY do

begin

GardenMas[I][J].CodGarden := 4;

GardenMas[I][J].Сulture.cod := 5;

// GardenMas[I][J].Сulture := 666;

end;

end;

for I := 0 to 6 do

begin

for J := 4 to \_GardenY do

begin

GardenMas[I][J].CodGarden := 4;

GardenMas[I][J].Сulture.cod := 1;

// GardenMas[I][J].Сulture := 666;

end;

end;

AssignFile(currFile, UserFileGarden);

Rewrite(currFile);

count := 0;

for I := 0 to \_GardenX do

begin

for J := 0 to \_GardenY do

begin

seek(currFile, count);

list := culturelist.Next;

while list <> nil do

begin

if list.culture.cod = GardenMas[I][J].Сulture.cod then

begin

GardenMas[I][J].Сulture.name := list.culture.name;

GardenMas[I][J].Сulture.Time := list.culture.Time;

break;

end;

list := list.Next;

end;

write(currFile, GardenMas[I][J]);

inc(count);

end;

end;

Closefile(currFile);

end;

procedure ReadFileColor;

var

currFile: file of RecordDictionary;

tempPair: RecordDictionary;

begin

AssignFile(currFile, FileColor);

Reset(currFile);

currMaxIdGarden := 0;

while not EoF(currFile) do

begin

Read(currFile, tempPair);

dictionaryColorToId.Add(tempPair.corGarden, tempPair.Color);

if currMaxIdGarden < tempPair.corGarden then

begin

currMaxIdGarden := tempPair.corGarden;

end;

end;

Close(currFile);

end;

procedure ReadGardenMas;

var

currFile: file of TGardenCell;

I, J: integer;

begin

AssignFile(currFile, UserFileGarden);

Reset(currFile);

for I := 0 to \_GardenX do

begin

for J := 0 to \_GardenY do

begin

Read(currFile, GardenMas[I][J]);

end;

end;

Closefile(currFile);

end;

end.

unit uAddNewCulture;

interface

uses

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

System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls, uConstData, uGLOBALdATA;

type

TForm3 = class(TForm)

Label1: TLabel;

ButtonCancel: TButton;

EditName: TEdit;

Label3: TLabel;

ButtonSave: TButton;

Label4: TLabel;

EditTime: TEdit;

procedure ButtonCancelClick(Sender: TObject);

procedure ButtonSaveClick(Sender: TObject);

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

private

culturelist: ptculture;

Res: TModalResult;

function IsValidInputData(const strName, strTime: string): boolean;

public

function ShowForAdd(var culture: cultureListInfo; culturelist: ptculture)

: TModalResult;

end;

var

Form3: TForm3;

implementation

{$R \*.dfm}

function TForm3.IsValidInputData(const strName, strTime: string): boolean;

function isNameCultureExist(const Name: string; list: ptculture): boolean;

begin

result := false;

if list = nil then

exit;

list := list.Next;

if list = nil then

exit;

while list <> nil do

begin

if list.culture.Name = Name then

begin

result := true;

exit;

end;

list := list.Next;

end;

end;

var

strmessage: string;

unusing: TMYDate;

begin

strmessage := '';

if not ConvertStringToDate(strTime, unusing) then

strmessage := 'Неверная дата' + #13#10;

if (StringReplace(strName, ' ', '', [rfReplaceAll]) = '') then

strmessage := strmessage + 'Неверное имя' + #13#10;

if isNameCultureExist(strName, culturelist) then

begin

strmessage := strmessage + 'Такой имя уже применяется' + #13#10;

end;

result := strmessage = '';

if (result) then

begin

end

else

begin

MessageBox(Application.Handle, PChar(strmessage), 'Ошибка ввода', MB\_OK);

end;

end;

function TForm3.ShowForAdd(var culture: cultureListInfo; culturelist: ptculture)

: TModalResult;

var

strmessage: string;

begin

Form3.culturelist := culturelist;

EditName.Text := '';

EditTime.Text := '';

result := ShowModal;

if Res = mrOk then

begin

inc(currMaxIdCulture);

culture.Name := EditName.Text;

culture.cod := currMaxIdCulture;

ConvertStringToDate(EditTime.Text, culture.Time);

// showMessage('индекс добавленого ' + inttostr(currMaxIdCulture));

end;

if Res = mrCancel then

begin

end;

result := Res;

end;

procedure TForm3.ButtonCancelClick(Sender: TObject);

begin

Res := mrnone;

Close();

end;

procedure TForm3.ButtonSaveClick(Sender: TObject);

begin

if IsValidInputData(EditName.Text, EditTime.Text) then

begin

Res := mrOk;

Close();

end;

end;

procedure TForm3.FormClose(Sender: TObject; var Action: TCloseAction);

begin

// Res := mrCancel;

end;

end.

unit uAddNewGarden;

interface

uses

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

System.Classes, Vcl.Graphics,

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

type

TForm4 = class(TForm)

EditName: TEdit;

Label1: TLabel;

Label2: TLabel;

Label3: TLabel;

ColorBox1: TColorBox;

ButtonSave: TButton;

ButtonCancel: TButton;

procedure ButtonSaveClick(Sender: TObject);

procedure ButtonCancelClick(Sender: TObject);

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

private

Res: TModalResult;

gardenlist: ptgarden;

function IsValidInputData(const strName: string;

const color: TColor): boolean;

public

function ShowForAdd(var garden: TGARDEN; gardenlist: ptgarden)

: TModalResult;

{ Public declarations }

end;

var

Form4: TForm4;

implementation

{$R \*.dfm}

function TForm4.IsValidInputData(const strName: string;

const color: TColor): boolean;

function IsColorAlreadyExist(const color: TColor): boolean;

begin

result := false;

for var Item in dictionaryColorToId.GetAllItems do

begin

if Item.Value = color then

begin

result := true;

end;

end;

end;

function isNameGardenExist(const Name: string; list: ptgarden): boolean;

begin

result := false;

if list = nil then

exit;

list := list.Next;

if list = nil then

exit;

while list <> nil do

begin

if list.garden.Name = Name then

begin

result := true;

exit;

end;

list := list.Next;

end;

end;

var

strmessage: string;

begin

strmessage := '';

if (StringReplace(strName, ' ', '', [rfReplaceAll]) = '') then

strmessage := strmessage + 'Неверно введено имя' + #13#10;

if IsColorAlreadyExist(color) then

strmessage := strmessage + 'Такой цвет уже применяется' + #13#10;

if isNameGardenExist(strName, gardenlist) then

begin

strmessage := strmessage + 'Такое имя уже применяется' + #13#10;

end;

result := strmessage = '';

if (result) then

begin

end

else

begin

MessageBox(Application.Handle, PChar(strmessage), 'Ошибка ввода', MB\_OK);

end;

end;

function TForm4.ShowForAdd(var garden: TGARDEN; gardenlist: ptgarden)

: TModalResult;

begin

Form4.gardenlist := gardenlist;

EditName.Text := '';

ColorBox1.Selected := clBlack;

result := ShowModal;

if Res = mrOk then

begin

inc(currMaxIdGarden);

garden.Name := EditName.Text;

garden.CodGarden := currMaxIdGarden;

// showMessage('индекс добавленого ' + inttostr(currMaxIdGarden));

dictionaryColorToId.Add(currMaxIdGarden, ColorBox1.Selected);

end;

if Res = mrCancel then

begin

end;

result := Res;

end;

procedure TForm4.ButtonCancelClick(Sender: TObject);

begin

Res := mrnone;

Close();

end;

procedure TForm4.ButtonSaveClick(Sender: TObject);

begin

if IsValidInputData(EditName.Text, ColorBox1.Selected) then

begin

Res := mrOk;

Close();

end;

end;

procedure TForm4.FormClose(Sender: TObject; var Action: TCloseAction);

begin

// Res := mrCancel;

end;

end.

unit uShowAllGardens;

interface

uses

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

System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.ExtCtrls, uGlobalData, uConstData,

Vcl.StdCtrls, Vcl.ComCtrls, udictionary;

type

TForm5 = class(TForm)

PaintBox1: TPaintBox;

ListViewGarden: TListView;

Label1: TLabel;

Button1: TButton;

Panel1: TPanel;

Label2: TLabel;

ListViewCulture: TListView;

Label3: TLabel;

ListViewCultureAll: TListView;

procedure PrintSelectedCulture(const nameCulture: string);

procedure CreateListViewGarden(gardenlist: PtGarden);

procedure PaintRect(const x, y: integer; const color: Tcolor;

const brush: Tcolor);

procedure PaintBox1Paint(Sender: TObject);

procedure FormShow(Sender: TObject);

procedure ListViewGardenClick(Sender: TObject);

procedure PrintSelectedGarden(const idGarden: integer);

procedure Button1Click(Sender: TObject);

procedure FormCreate(Sender: TObject);

procedure CreateListViewCulture(const idGarden: array of integer);

procedure PaintBox1MouseDown(Sender: TObject; Button: TMouseButton;

Shift: TShiftState; x, y: integer);

procedure CreateListCultureAll(CultutrList: ptCulture);

procedure ListViewCultureAllClick(Sender: TObject);

procedure printAllGarden;

private

culturelist: ptCulture;

gardenlist: PtGarden;

public

function FormShowAll(culturelist: ptCulture; gardenlist: PtGarden)

: TModalResult;

{ Public declarations }

end;

var

Form5: TForm5;

const

SizeRect = 20;

SizePen = 2;

implementation

{$R \*.dfm}

procedure TForm5.CreateListViewGarden(gardenlist: PtGarden);

var

listitem: tlistitem;

strColor: string;

begin

ListViewGarden.Clear;

gardenlist := gardenlist.Next;

while gardenlist <> nil do

begin

listitem := ListViewGarden.Items.Add;

listitem.Caption := gardenlist.garden.Name;

strColor := ColorToString(IdentifyColor(gardenlist.garden.CodGarden));

listitem.SubItems.Add(Copy(strColor, 3, length(strColor)));

gardenlist := gardenlist.Next;

end;

end;

procedure TForm5.printAllGarden;

begin

PaintBox1.Canvas.brush.color := clBtnFace;

PaintBox1.Canvas.FillRect(PaintBox1.ClientRect);

for var i := 0 to \_GardenX do

begin

for var j := 0 to \_GardenY do

begin

PaintRect(i, j, clsilver, IdentifyColor(gardenmas[i][j].CodGarden));

end;

end;

end;

procedure TForm5.CreateListCultureAll(CultutrList: ptCulture);

var

Column: TListColumn;

listitem: tlistitem;

begin

ListViewCultureAll.Clear();

CultutrList := CultutrList.Next;

while CultutrList <> nil do

begin

listitem := ListViewCultureAll.Items.Add;

listitem.Caption := CultutrList.culture.Name;

CultutrList := CultutrList.Next;

end;

end;

procedure TForm5.PrintSelectedCulture(const nameCulture: string);

var

Rect: TRect;

begin

printAllGarden;

for var i := 0 to \_GardenX do

begin

for var j := 0 to \_GardenY do

begin

if gardenmas[i][j].Сulture.Name = nameCulture then

begin

PaintBox1.Canvas.brush.color := NegativeColor(IdentifyColor(gardenmas[i][j].CodGarden));

PaintBox1.Canvas.Pen.color := IdentifyColor(gardenmas[i][j].CodGarden);

Rect.Create(i \* SizeRect + SizePen, j \* SizeRect + SizePen,

(i) \* SizeRect + SizeRect \* 2 div 3, (j) \* SizeRect + SizeRect

\* 2 div 3);

PaintBox1.Canvas.Rectangle(Rect);

// PaintRect(i, j, IdentifyColor(GardenMas[i][j].CodGarden), clsilver);

end;

end;

end;

end;

procedure TForm5.PrintSelectedGarden(const idGarden: integer);

begin

// PaintBox1.Canvas.brush.color := clBtnFace;

// PaintBox1.Canvas.FillRect(PaintBox1.ClientRect);

for var i := 0 to \_GardenX do

begin

for var j := 0 to \_GardenY do

begin

PaintRect(i, j, IdentifyColor(gardenmas[i][j].CodGarden), clsilver);

end;

end;

for var i := 0 to \_GardenX do

begin

for var j := 0 to \_GardenY do

begin

if gardenmas[i][j].CodGarden = idGarden then

begin

PaintRect(i, j, clsilver, IdentifyColor(gardenmas[i][j].CodGarden));

end;

end;

end;

end;

procedure TForm5.Button1Click(Sender: TObject);

var

MySet: array of integer;

function IsCodExist(const digits: integer): boolean;

begin

Result := false;

for var item in MySet do

begin

if item = digits then

begin

Result := true;

break;

end;

end;

end;

begin

for var i := 0 to \_GardenX do

begin

for var j := 0 to \_GardenY do

begin

PaintRect(i, j, clsilver, IdentifyColor(gardenmas[i][j].CodGarden));

if not IsCodExist(gardenmas[i][j].CodGarden) then

begin

setLength(MySet, length(MySet) + 1);

MySet[length(MySet) - 1] := gardenmas[i][j].CodGarden;

end;

end;

end;

ListViewCulture.Clear;

CreateListViewCulture(MySet);

end;

procedure TForm5.CreateListViewCulture(const idGarden: array of integer);

function getNameCulture(const id: integer; list: ptCulture): string;

begin

list := list.Next;

while list <> nil do

begin

if list.culture.cod = id then

begin

Result := list.culture.Name;

exit;

end;

list := list.Next;

end;

end;

var

FrequensChars: TMyDictionary<integer, integer>;

listitem: tlistitem;

begin

FrequensChars := TMyDictionary<integer, integer>.Create;

for var k := Low(idGarden) to High(idGarden) do

begin

for var i := 0 to \_GardenX do

begin

for var j := 0 to \_GardenY do

begin

if gardenmas[i][j].CodGarden = idGarden[k] then

begin

if FrequensChars.ContainsKey(gardenmas[i][j].Сulture.cod) then

begin

FrequensChars[gardenmas[i][j].Сulture.cod] :=

FrequensChars[gardenmas[i][j].Сulture.cod] + 1;

end

else

begin

FrequensChars.Add(gardenmas[i][j].Сulture.cod, 1);

end;

// showmessage(inttostr(idGarden[k]) + ' : ' + inttostr(FrequensChars[gardenmas[i][j].Сulture.cod]));

end;

end;

end

end;

for var item in FrequensChars.GetAllItems do

begin

listitem := ListViewCulture.Items.Add;

listitem.Caption := getNameCulture(item.Key, culturelist);

listitem.SubItems.Add(inttostr(item.Value));

end;

end;

procedure TForm5.FormCreate(Sender: TObject);

var

Column: TListColumn;

x: integer;

begin

Column := ListViewGarden.Columns.Add;

Column.Caption := 'Грядка';

Column := ListViewGarden.Columns.Add;

Column.Caption := 'Цвет';

ListViewGarden.Columns[0].Width := 150;

ListViewGarden.Columns[1].Width := 150;

Column := ListViewCulture.Columns.Add;

Column.Caption := 'Культура';

Column := ListViewCulture.Columns.Add;

Column.Caption := 'Кол-во';

ListViewCulture.Columns[0].Width := 80;

ListViewCulture.Columns[1].Width := 80;

Column := ListViewCultureAll.Columns.Add;

Column.Caption := 'Культура';

ListViewCultureAll.Columns[0].Width := 150;

Panel1.color := clCream;

PaintBox1.Width := (1 + \_GardenX) \* (SizeRect);

PaintBox1.Height := (1 + \_GardenY) \* (SizeRect);

Panel1.Width := PaintBox1.Width + 2 \* 7;

Panel1.Height := PaintBox1.Height + 2 \* 7;

PaintBox1.Left := 7;

PaintBox1.Top := 7;

end;

procedure TForm5.FormShow(Sender: TObject);

begin

ListViewCulture.Clear;

ListViewGarden.Clear;

CreateListViewGarden(gardenlist);

CreateListCultureAll(culturelist);

Button1.Click();

end;

function TForm5.FormShowAll(culturelist: ptCulture; gardenlist: PtGarden)

: TModalResult;

begin

Form5.culturelist := culturelist;

Form5.gardenlist := gardenlist;

Result := ShowModal;

CreateListCultureAll(culturelist);

end;

procedure TForm5.ListViewCultureAllClick(Sender: TObject);

var

SelectedItem: tlistitem;

begin

SelectedItem := ListViewCultureAll.Selected;

if SelectedItem <> nil then

begin

if Assigned(SelectedItem) then

begin

PrintSelectedCulture(SelectedItem.Caption);

end;

end;

end;

procedure TForm5.ListViewGardenClick(Sender: TObject);

var

SelectedItem: tlistitem;

begin

SelectedItem := ListViewGarden.Selected;

if SelectedItem <> nil then

begin

if Assigned(SelectedItem) then

begin

PrintSelectedGarden(getIdGarden(SelectedItem.Caption, gardenlist));

end;

ListViewCulture.Clear;

CreateListViewCulture(getIdGarden(SelectedItem.Caption, gardenlist));

end;

end;

procedure TForm5.PaintRect(const x, y: integer; const color: Tcolor;

const brush: Tcolor);

var

Rect: TRect;

begin

PaintBox1.Canvas.brush.color := brush;

PaintBox1.Canvas.Pen.color := color;

Rect.Create(x \* SizeRect + SizePen, y \* SizeRect + SizePen,

(x + 1) \* SizeRect, (y + 1) \* SizeRect);

PaintBox1.Canvas.Rectangle(Rect);

end;

procedure TForm5.PaintBox1MouseDown(Sender: TObject; Button: TMouseButton;

Shift: TShiftState; x, y: integer);

var

idGarden: integer;

begin

idGarden := gardenmas[x div SizeRect][y div SizeRect].CodGarden;

PrintSelectedGarden(idGarden);

ListViewCulture.Clear;

CreateListViewCulture(idGarden);

end;

procedure TForm5.PaintBox1Paint(Sender: TObject);

begin

printAllGarden;

end;

end.

unit uDictionary;

interface

uses Vcl.Dialogs;

type

TKeyValuePair<TKey, TValue> = record

Key: TKey;

Value: TValue;

end;

TMyDictionary<TKey, TValue> = class

private

FItems: array of TKeyValuePair<TKey, TValue>;

function FindIndex(const Key: TKey): Integer;

public

procedure Add(const Key: TKey; const Value: TValue);

function ContainsKey(const Key: TKey): Boolean;

function GetValue(const Key: TKey): TValue;

procedure Remove(const Key: TKey);

function GetAllItems: TArray<TKeyValuePair<TKey, TValue>>;

property Items[const Key: TKey]: TValue read GetValue write Add; default;

end;

implementation

procedure TMyDictionary<TKey, TValue>.Add(const Key: TKey; const Value: TValue);

var

Index: Integer;

begin

Index := FindIndex(Key);

if Index >= 0 then

FItems[Index].Value := Value

else

begin

SetLength(FItems, Length(FItems) + 1);

FItems[High(FItems)].Key := Key;

FItems[High(FItems)].Value := Value;

end;

end;

function TMyDictionary<TKey, TValue>.ContainsKey(const Key: TKey): Boolean;

begin

Result := FindIndex(Key) >= 0;

end;

function TMyDictionary<TKey, TValue>.FindIndex(const Key: TKey): Integer;

var

I: Integer;

begin

for I := 0 to high(FItems) do

if FItems[I].Key = (Key) then

begin

Result := I;

Exit();

end;

Result := -1;

end;

function TMyDictionary<TKey, TValue>.GetAllItems

: TArray<TKeyValuePair<TKey, TValue>>;

var

I: Integer;

begin

SetLength(Result, Length(FItems));

for I := 0 to High(FItems) do

Result[I] := FItems[I];

end;

function TMyDictionary<TKey, TValue>.GetValue(const Key: TKey): TValue;

var

Index: Integer;

begin

Index := FindIndex(Key);

if Index >= 0 then

Result := FItems[Index].Value

end;

procedure TMyDictionary<TKey, TValue>.Remove(const Key: TKey);

var

Index, I: Integer;

begin

Index := FindIndex(Key);

if Index = -1 then

showmessage('Ex : not found key');

if Index >= 0 then

begin

for I := Index to High(FItems) - 1 do

FItems[I] := FItems[I + 1];

SetLength(FItems, Length(FItems) - 1);

end;

end;

end.

unit uSP;

interface

uses

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

System.Classes, Vcl.Graphics,

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

type

TForm6 = class(TForm)

Label1: TLabel;

Button1: TButton;

procedure Button1Click(Sender: TObject);

procedure FormCreate(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

Form6: TForm6;

implementation

{$R \*.dfm}

procedure TForm6.Button1Click(Sender: TObject);

begin

Close();

end;

procedure TForm6.FormCreate(Sender: TObject);

begin

Label1.Caption := textSP;

end;

end.

unit uWorks;

interface

uses

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

System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.ExtCtrls, Vcl.WinXCtrls,

Vcl.ComCtrls, Vcl.StdCtrls, uglobaldata, uaddwork;

type

TForm7 = class(TForm)

SplitView1: TSplitView;

ListView1: TListView;

SplitView2: TSplitView;

ButtonAdd: TButton;

ButtonDelete: TButton;

procedure FormCreate(Sender: TObject);

procedure FormShow(Sender: TObject);

procedure createListView1(list: ptworks);

procedure ButtonAddClick(Sender: TObject);

procedure ListView1Click(Sender: TObject);

procedure ButtonDeleteClick(Sender: TObject);

private

listWorks: ptworks;

procedure Add(const newwork: worksRecord);

procedure Delete(const name: string);

{ Private declarations }

public

function MyShow(list: ptworks): TModalResult;

{ Public declarations }

end;

var

Form7: TForm7;

implementation

{$R \*.dfm}

procedure TForm7.ButtonAddClick(Sender: TObject);

var

work: worksRecord;

begin

work.name := '';

work.Description := '';

work.Date := '';

if Form8.ShowForEditing(work, ListView1) = mrok then

begin

Add(work);

end;

ButtonDelete.Enabled := ListView1.Selected <> nil;

end;

procedure TForm7.Add(const newwork: worksRecord);

var

firstNode: ptworks;

begin

firstNode := listWorks.next;

new(listWorks.next);

listWorks.next.work := newwork;

listWorks.next.next := firstNode;

createListView1(listWorks);

end;

procedure TForm7.createListView1(list: ptworks);

var

Column: TListColumn;

listitem: TListItem;

begin

ListView1.Clear();

list := list.next;

while list <> nil do

begin

listitem := ListView1.Items.Add;

listitem.Caption := '"' + list.work.name + '"';

listitem.SubItems.Add(list.work.Description);

listitem.SubItems.Add(list.work.Date);

list := list.next;

end;

end;

function TForm7.MyShow(list: ptworks): TModalResult;

begin

listWorks := list;

result := showmodal;

createListView1(listWorks);

end;

procedure TForm7.Delete(const name: string);

function CompareStrings(const Str1, Str2: string): Boolean;

begin

Result := Str1 = Str2;

end;

var

list: ptworks;

begin

list := listWorks;

while (list.next <> nil) and (not CompareStrings( list.next.work.name , name)) do

begin

list := list.next;

end;

if (list.next <> nil) and ( CompareStrings( list.next.work.name , name)) then

begin

list.next := list.next.next;

end;

end;

procedure TForm7.ButtonDeleteClick(Sender: TObject);

function RemoveFirstAndLastChar(const Input: string): string;

begin

if Length(Input) <= 2 then

Result := ''

else

Result := Copy(Input, 2, Length(Input) - 2);

end;

var

SelectedItem: TListItem;

begin

if ListView1.Selected <> nil then

begin

SelectedItem := ListView1.Selected;

if Assigned(SelectedItem) then

begin

Delete(RemoveFirstAndLastChar(SelectedItem.Caption));

createListView1(listWorks);

end;

end

else

begin

ShowMessage('Ex');

end;

ButtonDelete.Enabled := ListView1.Selected <> nil;

end;

procedure TForm7.FormCreate(Sender: TObject);

var

Column: TListColumn;

begin

Column := ListView1.Columns.Add;

Column.Caption := 'Название дела';

Column := ListView1.Columns.Add;

Column.Caption := 'Описание';

Column := ListView1.Columns.Add;

Column.Caption := 'Запланированная дата';

ListView1.Columns[0].Width := 150;

ListView1.Columns[1].Width := 150;

ListView1.Columns[2].Width := 250;

new(listWorks);

end;

procedure TForm7.FormShow(Sender: TObject);

begin

ButtonDelete.Enabled := ListView1.Selected <> nil;

createListView1(listWorks);

end;

procedure TForm7.ListView1Click(Sender: TObject);

begin

ButtonDelete.Enabled := ListView1.Selected <> nil;

end;

end.

unit uAddWork;

interface

uses

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

System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.ExtCtrls, Vcl.WinXCtrls,

Vcl.ComCtrls, Vcl.StdCtrls, uglobaldata;

type

TForm8 = class(TForm)

EditName: TEdit;

EditDesc: TEdit;

Editdate: TEdit;

Label1: TLabel;

Label2: TLabel;

Label3: TLabel;

ButtonOk: TButton;

Buttoncancel: TButton;

procedure ButtonOkClick(Sender: TObject);

procedure ButtonCancelClick(Sender: TObject);

private

masNames: array of string;

res: TModalResult;

function IsValidInputData(const str1: string): boolean;

{ Private declarations }

public

function ShowForEditing(var Task: worksRecord; ListView1: TLISTVIEW)

: TModalResult;

end;

var

Form8: TForm8;

implementation

{$R \*.dfm}

function TForm8.ShowForEditing(var Task: worksRecord; ListView1: TLISTVIEW)

: TModalResult;

begin

setlength(masNames, ListView1.Items.Count);

for var i := 0 to ListView1.Items.Count - 1 do

masNames[i] := ListView1.Items[i].Caption;

EditName.Text := Task.Name;

EditDesc.Text := Task.Description;

Editdate.Text := Task.Date;

result := ShowModal;

if res = mrOk then

begin

Task.Name := EditName.Text;

Task.Description := EditDesc.Text;;

Task.Date := Editdate.Text;

end;

result := res;

end;

function TForm8.IsValidInputData(const str1: string): boolean;

begin

if '' = str1 then

begin

result := false;

exit;

end;

for var item in masNames do

begin

if item = '"' + str1 + '"' then

begin

result := false;

exit;

end;

end;

result := True;

end;

procedure TForm8.ButtonCancelClick(Sender: TObject);

begin

res := mrnone;

Close();

end;

procedure TForm8.ButtonOkClick(Sender: TObject);

begin

if IsValidInputData(EditName.Text) then

begin

res := mrOk;

Close();

end

else

begin

MessageBox(Application.Handle, 'Неверно введённое имя '+#13#10 + '(название не может быть пустым)'+#13#10 + '(название должно быть уникальным)', 'Ошибка ввода', MB\_OK);

end;

end;

end.