Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Основы алгоритмизации и программирования (ОАиП)

**ОТЧЕТ**

по учебной практике

Вариант 3

Студент гр. 351004 Баражалович А.И.

Руководитель Данилова Г.В.

Минск 2024

СОДЕРЖАНИЕ

[Введение 3](#_Toc136223397)

[1 Постановка задачи 4](#_Toc136223398)

[2 Проектирование программного средства 5](#_Toc136223399)

[2.1 Структура программы 5](#_Toc136223400)

[2.2 Проектирование интерфейса программного средства 6](#_Toc136223401)

[2.3 Проектирование функционала программного средства 10](#_Toc136223402)

[3 Разработка программного средства 15](#_Toc136223403)

[3.1 Взаимодействие со списками 15](#_Toc136223407)

[3.2 Взаимодействие с таблицей 16](#_Toc136223408)

[3.3 Работа с файлами 17](#_Toc136223409)

[4 Тестирование программного средства 20](#_Toc136223410)

[5 Руководство пользователя 18](#_Toc136223411)

[5.1 Интерфейс программного средства 18](#_Toc136223412)

[5.2 Управление программным средством 26](#_Toc136223413)

[Заключение 27](#_Toc136223414)

[Список использованных источников 28](#_Toc136223415)

[Приложение A. Текст программы 29](#_Toc136223416)

# ВВЕДЕНИЕ

Память является ценным ресурсом в вычислительных системах, и программисту необходимо эффективно управлять ею. Для этого существуют два основных подхода к распределению памяти: статическое и динамическое.

Статическое распределение памяти происходит на этапе компиляции или трансляции программы. Здесь память заранее выделяется для переменных и данных, и они остаются неизменными в течение выполнения программы. Это позволяет достичь высокой производительности, поскольку нет необходимости в дополнительных операциях управления памятью во время выполнения программы. Однако этот подход требует заранее известного объема памяти, что может быть недостаточно гибким для некоторых задач.

В отличие от статического подхода, динамическое управление памятью осуществляется во время выполнения программы. Здесь память выделяется и освобождается по мере необходимости. Это позволяет более гибко использовать память и адаптироваться к изменяющимся требованиям программы. Например, когда требуется работа с переменным объемом данных или при создании структур данных, которые могут изменяться в размере.

Для организации динамической памяти используются указатели или ссылочные типы данных. Указатель представляет собой переменную, которая содержит адрес области памяти, где хранятся данные определенного типа. Указатели могут быть типизированными, то есть связанными с определенным типом данных, чтобы обеспечить безопасность и правильное обращение к данным. Область памяти для хранения указателей выделяется статически, в то время как память для переменных, на которые указывают указатели, выделяется динамически по мере необходимости.

Динамическое распределение памяти широко применяется в различных областях, включая обработку графики, звука, баз данных и других задач, где требуется гибкое использование памяти во время выполнения программы. Однако при использовании динамической памяти программисту также необходимо следить за освобождением уже ненужных участков памяти, чтобы избежать утечек памяти и эффективно использовать ресурсы системы.

Целью данной учебной практики является разработка программного средства “Электронная книга студентов” с использованием динамических структур.

# ПОСТАНОВКА ЗАДАЧИ

Необходимо разработать приложение, которое позволит хранить и управлять информацией о студентах в группах университета.

Запись списка групп содержит следующие поля:

* номер группы;
* год образования;
* код специальности;
* число студентов.

Запись списка студентов содержит следующие поля:

* номер группы;
* ФИО студента;
* оценки за последнюю сессию;
* средний балл.

Функциональные требования:

* cформировать список студентов-задолжников по результатам сессии по каждой дисциплине отдельно;
* cформировать список студентов для отчисления (получивших 3 и более неудовлетворительных оценки);
* осуществлять поиск студента в исходном списке по ФИО;
* предусмотреть возможность добавлять, удалять и корректировать записи из исходного списка, а также просматривать весь список.

Для разработки программного средства будет использоваться язык программирования Delphi и среда разработки Embarcadero Delphi 11.3 Community Edition.

**2 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

**2.1 Структура программы**

При разработке приложения будет использовано двенадцать модулей:

* MainMenuUnit – модуль, отвечающий за отображение главного меню приложения. Он предоставляет пользователю список доступных функций и действий, которые можно выполнить в приложении.
* ViewGroupList – модуль, ответственный за отображение окна со списком групп. Он позволяет пользователю просматривать информацию о группах.
* VeiwStudentsUnit– модуль, ответственный за отображение окна со списком группы. Он позволяет пользователю просматривать информацию о студентах в группе.
* FindStudentUnit– модуль, обеспечивающий поиск полной информации о студенте. Он позволяет пользователю искать и получать подробную информацию о студенте.
* GroupListActionUnit – модуль, позволяющий пользователю добавлять новые записи в список групп. Он обеспечивает возможность ввода данных о новых группах.
* StudentsInGroupListActionUnit – модуль, позволяющий пользователю добавлять новые записи в список студентов. Он обеспечивает возможность ввода данных о новых студентах в группе.
* GroupsLinkedListUnit – модуль, содержащий реализацию однонаправленного списка для работы с группами. Он предоставляет функциональность для управления списком групп.
* StudentLinkedListUnit – модуль, содержащий реализацию однонаправленного списка для работы со студентами. Он предоставляет функциональность для управления списком студентов.
* ListsActionUnit – модуль, обеспечивающий корректный вывод информации о студентах во время использования специальных функций.
* FileUnit – модуль, обеспечивающий функциональность работы с файлами. Он позволяет пользователю открывать и сохранять файлы, связанные с группами, студентами.
* InputEditUnit – модуль, который обеспечивает корректный ввод информации пользователем. Он проверяет введенные значения на соответствие требованиям положительных чисел и букв алфавита. Предоставляет возможность ввода только допустимых значений.

**2.2 Проектирование интерфейса программного средства**

Внешний вид программных средств играет существенную роль в обеспечении удобства пользователей и качества самого средства.

Интерфейс должен удовлетворять следующим требованиям:

* простота и понятность в использовании;
* удобное расположение функций.

**2.2.1** Главное окно

Главное окно приложения состоит из четырех кнопок, с помощью которых пользователь может переходить между окнами программного средства:

* первая кнопка открывает модуль “ViewGroupListUnit”, где пользователь может просмотреть список групп;
* вторая кнопка позволяет перейти в модуль “FindStudentUnit”, где пользователь может найти студента в списке по ФИО;
* третья кнопка, названная “Список студентов-задолжников”, позволяет пользователю вывести список студентов-задолжников, у которых есть неудовлетворительные оценки;
* четвертая кнопка, названная “Список студентов на отчисление”, позволяет пользователю вывести список студентов, у которых есть более трех неудовлетворительных оценок.

Макет главного окна приложения представлен на рисунке 2.1.



Рисунок 2.1 – Макет главного окна приложения

**2.2.2** Окно просмотра списка групп

Окно просмотра списка групп состоит из кнопок и списка. В верхней части формы находится строка с кнопками:

* добавить новую группу;
* удалить выбранную группу;
* редактировать выбранную группу;
* просмотр состава группы.

Отображение списка групп осуществляются с помощью компонента TStringGrid. В таблице групп будут присутствовать такие поля, как:

* номер группы;
* год образования группы;
* код группы;
* число студентов в группе.

Макет окна просмотра списка групп представлен на рисунке 2.2.

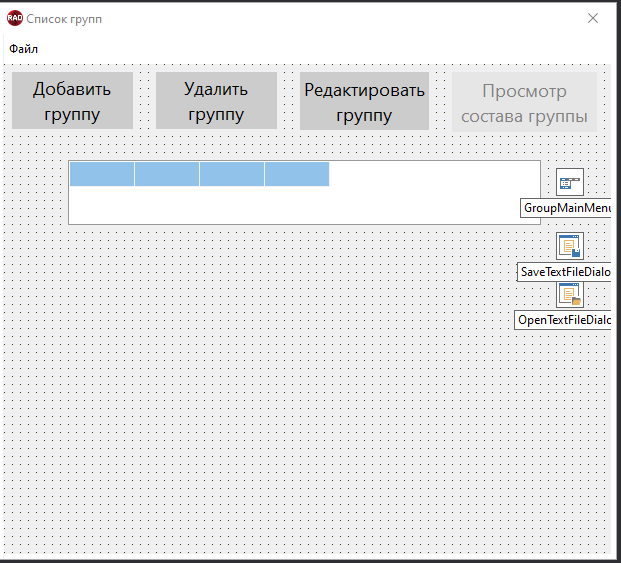


Рисунок 2.2 – Макет окна просмотра списка групп

**2.2.3** Окно добавления группы

Окно добавления группы состоит из нескольких однострочных полей ввода и двух кнопок. Выбор подтверждается нажатием на компонент TButton. Макет окна добавления группы представлен на рисунке 2.3.

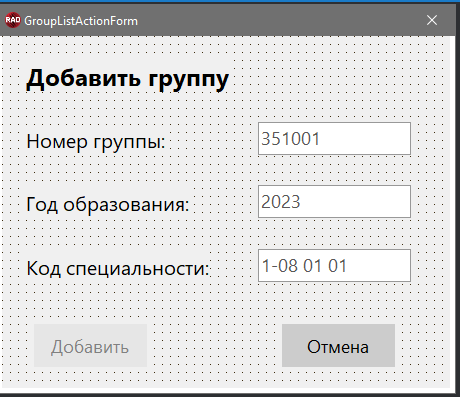


Рисунок 2.3 – Макет окна добавления группы

**2.2.4** Окно просмотра списка студентов в группе

Окно просмотра списка студентов в группе состоит из кнопок и списка. В верхней части формы находится строка с кнопками:

* добавить нового студента;
* редактировать выбранного студента;
* удалить выбранного студента.

Отображение списка студентов в группе осуществляются с помощью компонента TStringGrid. В таблице студентов будут присутствовать такие поля, как:

* номер группы;
* ФИО студента;
* оценки за последнюю сессию;
* средний балл.

Макет окна просмотра списка студентов в группе представлен на рисунке 2.4.

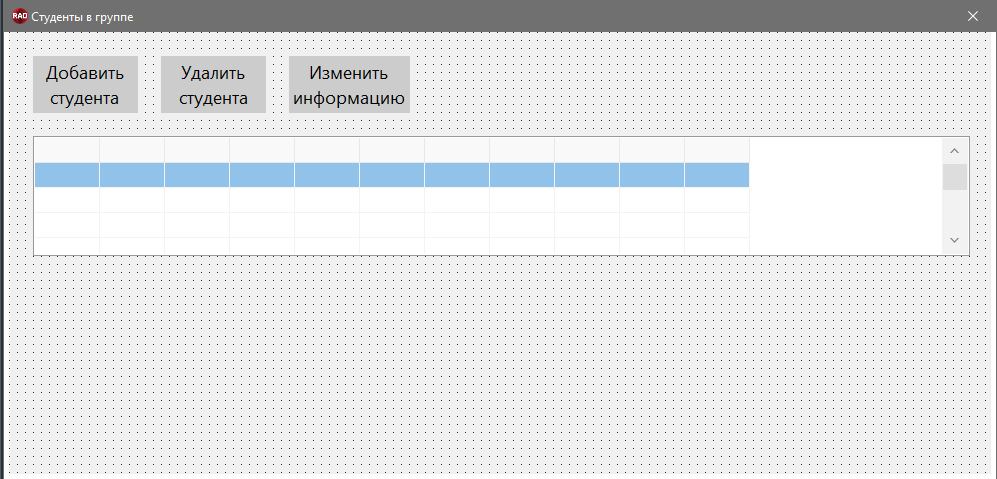


Рисунок 2.4 – Макет окна просмотра списка студентов в группе

**2.2.5** Окно добавления студента в группу

Окно добавления студента в группу состоит из нескольких однострочных полей ввода, ячеек для ввода оценок и двух кнопок. Выбор подтверждается нажатием на компонент TButton. Макет окна добавления студента в группу представлен на рисунке 2.5.

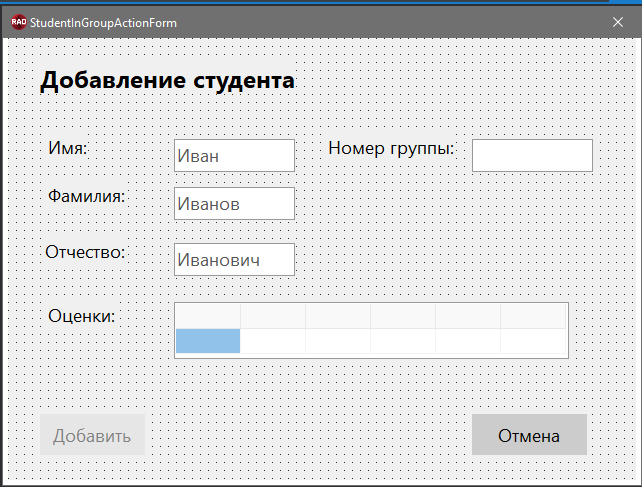


Рисунок 2.5 – Макет окна добавления студента в группу

**2.2.6** Окно поиска студента

Окно поиска состоит из полей для ввода ФИО, а также кнопки для поиска. Макет окна поиска студента на рисунке 2.6.

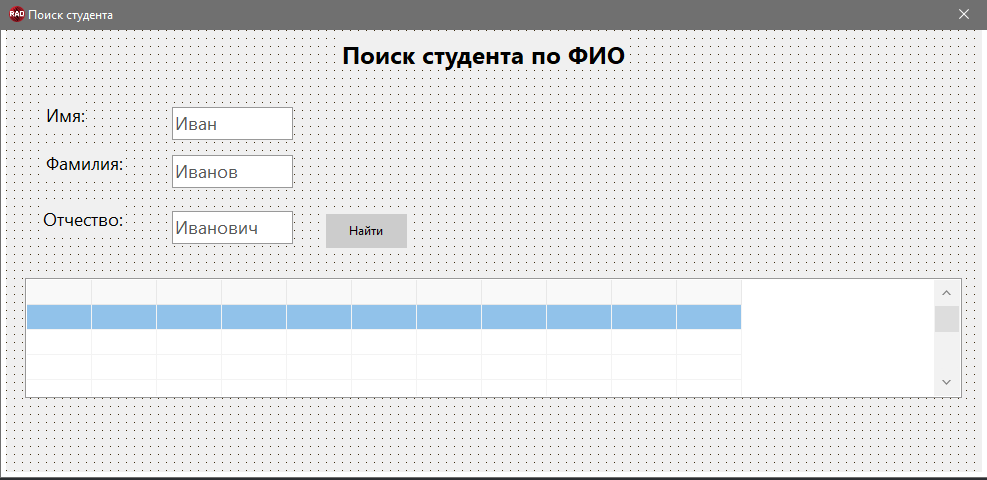


Рисунок 2.6 – Макет окна поиска студнта

**2.2.7** Окно формирования списка студентов-задолжников

Окно формирования списка состоит из выпадающего списка с дисциплинами и поля для отображения информации о студентах. Макет окна формирования списка студентов-задолжников на рисунке 2.7.

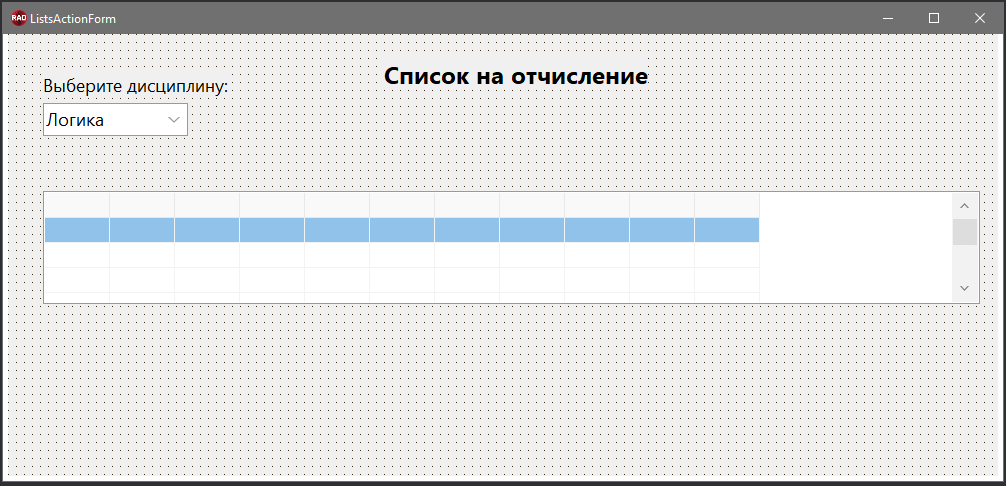


Рисунок 2.7 – Макет окна формирования списка студентов-задолжников

**2.3 Проектирование функционала программного средства**

Для работы программы был выбран тип данных “однонаправленный список”.

Для решения поставленной задачи потребуется разработать несколько алгоритмов, реализующих основные функции программы:

* поиск полной информации о студенте;
* формирование списка студентов-должников;
* добавление студента;
* сохранение списка в файл.

**2.3.1** Поиск полной информации о студенте

Для получения полной информации о студенте по ФИО необходимо просмотреть список студентов от начала до конца и сравнивать ФИО каждого студента с искомым. Блок-схема функции нахождения полной информации студента по ФИО.



Рисунок 2.8 – Блок-схема функции SearchStudentNSP

**2.3.2** Формирование списка студентов на отчисление

Для формирования списка студентов на отчисление первоначально создается пустой список. Затем он наполняется студентами, используя функцию-фильтр, которая производит поиск неудовлетворительных оценок в количестве более 3-х штук. Блок-схема функции для формирования списка студентов на отчисление представлена ниже.



Рисунок 2.9 – Блок-схема функции FindExpellStudents

**2.3.3** Добавление записи в список

Для добавления записи о кандидате будет использоваться процедура, которая будет создавать новую запись о студенте, основываясь на переданном ей параметре, и помещать запись в конец списка. Блок-схема процедуры для добавления студента представлена ниже.



Рисунок 2.10 – Блок-схема процедуры AddStudent

**2.3.4** Сохранение списков

Процедура сохранения списка групп и студентов записывает данные о каждой группе и студенте в типизированные файлы, до того момента, пока указатель на следующую запись будет задан определенным значением. Блок-схема сохранения списков групп и студентов представлена ниже.



Рисунок 2.11 – Блок-схема процедуры WriteFileData

Если файл с именем, переданным в качестве параметра в процедуру, не был найден, то данная процедура создает его и производит запись.

# РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА



## **Взаимодействие со списками**

Для хранения информации о студентах будет использоваться динамическая структура данных однонаправленный список.

**3.1.1** Добавление записи в список

Для добавления записи о студенте будет использоваться процедура, которая будет создавать новую запись о студенте, основываясь на переданном ей параметре, и помещать запись в конец списка. Код процедуры для добавления студенте представлен ниже.

Procedure AddStudent(Var StudentList: PStudent; Const StudentData: TStudentData);

Var

NewStudent, CurrStudent: PStudent;

Begin

// Создание нового студента

NewStudent := CreateStudent(StudentData);

If StudentList = Nil Then

StudentList := NewStudent

// Eсли список до этого имел элементы

Else

Begin

// Перемещение указателя в конец списка

CurrStudent := StudentList;

While CurrStudent^.Next <> Nil Do

CurrStudent := CurrStudent^.Next;

// Добавление нового студента в конец списка

CurrStudent^.Next := NewStudent;

End;

End;

**3.1.2** Удаление списка студентов

Для очистки списка студентов будет использоваться процедура ClearStudents. Код процедуры для удаления списка представлен ниже.

Procedure ClearStudents(Var StudentList: PStudent);

Var

CurrStudent, TempStudent: Pstudent;

Begin

// Получение указателя на начало списка

CurrStudent := StudentList;

// Проход по всему списку

While CurrStudent <> Nil Do

Begin

// Срхранение удаляемого студента

TempStudent := CurrStudent;

// Создание новой связи

CurrStudent := CurrStudent^.Next;

// Очищение памяти

Dispose(TempStudent);

End;

// Разрывание связи указателя списка

StudentList := Nil;

End;

## Взаимодействие с таблицей

Взаимодействие с таблицей производится после вызова модального окна и изменения списка групп.

**3.2.1** Добавление новой строки

Для создания новой записи о группе необходимо использовать метод AddGroupClick, который будет вызывать открытие специальной формы для заполнения данных. Код метода для добавления новой строки в таблицу представлен ниже.

Procedure TGroupListForm.AddGroupClick(Sender: TObject);

Begin

// Установление типа формы

StateOfGForm := AddG;

// Создание формы для добавления группы

CreateActionForm('Добавление группы', 'Добавить');

// Нажата ли кнопка добавить

If IsEdited Then

Begin

// Увеличение числа строк на 1

GroupGrid.RowCount := GroupGrid.RowCount + 1;

// Заполнение таблицы

DrawGroupGrid(GroupGrid, GroupList);

IsEdited := False;

// Изменение параметров сохранения

IsSaved := False;

SaveFile.Enabled := True;

End;

End;

**3.2.2** Удаление строки

Для удаления информации о группе необходимо использовать метод DeleteGroupClick. Код метода для удаления строки в таблице представлен ниже.

Procedure TGroupListForm.DeleteGroupClick(Sender: TObject);

Var

Confirmation, GroupCount: Integer;

Begin

// Выбран редактируемый ряд

If GroupGrid.Row > 0 Then

Begin

// Предупреждение о удалении =

Confirmation := Application.MessageBox('Вы действительно хотите удалить группу?', 'Удаление кандидата', MB\_YESNO + MB\_ICONQUESTION + MB\_DEFBUTTON2);

If Confirmation = IDYES Then

Begin

// Удаление группы

DeleteGroupF(GroupList, GetGroupDataFromGrid(GroupGrid), StudentsList);

// Уменьшение количества строк на 1

GroupGrid.RowCount := GroupGrid.RowCount - 1;

// Заполнение таблицы

DrawGroupGrid(GroupGrid, GroupList);

// Подсчёт количества групп

GroupCount := CountGroups(GroupList);

// Изменение параметров сохранения

IsSaved := GroupCount = 0;

SaveFile.Enabled := GroupCount > 0;

End;

End

Else

// Сообщение об ошибке

Application.MessageBox('Не выбрано редактируемое поле!',

'Предупреждение', MB\_OK + MB\_ICONWARNING);

End;

## **Работа с файлами**

В данной программе, как и было упомянуто ранее, предусмотрена пользовательская работа с файлами, а именно, загрузка и сохранение данных. Функция загрузки файлов позволяет пользователю выбрать файл на своем компьютере и загрузить его в программу.

**3.3.1** Сохранение списков в файл

Для сохранения списков в файл используется метод SaveFileClick.

Код метода SaveFileClick представлен ниже.

Procedure TGroupListForm.SaveFileClick(Sender: TObject);

Var

OutputGroupFile: TGroupDataFile;

OutputStudentFile: TStudentDataFile;

Begin

// Изменение базового расширения файла при открытии диалогового окна сохранения групп

SaveTextFileDialog.DefaultExt := 'grplst';

// Диалоговое окно сохранения файла групп

If SaveTextFileDialog.Execute Then

Begin

AssignFile(OutputGroupFile, SaveTextFileDialog.FileName);

// Вызов функции записи в файл групп

WriteFileData(OutputGroupFile, OutputStudentFile, GroupList,

StudentsList, True);

End;

// Изменение базового расширения файла при открытии диалогового окна сохранения студентов

SaveTextFileDialog.DefaultExt := 'stdlst';

// Диалоговое окно сохранения файла студентов

If SaveTextFileDialog.Execute Then

Begin

AssignFile(OutputStudentFile, SaveTextFileDialog.FileName);

// Вызов функции записи в файл студентов

WriteFileData(OutputGroupFile, OutputStudentFile, GroupList,

StudentsList, False);

// Результат сохранён

IsSaved := True;

End;

End;

**3.3.2** Добавление записей в списки из файла

Для добавления записей в списки из файла используется процедура OpenFileClick. Код процедуры OpenFileClick представлен ниже.

Procedure TGroupListForm.OpenFileClick(Sender: TObject);

Var

InputGroupFile: TGroupDataFile;

InputStudentFile: TStudentDataFile;

TempGroupList: PGroup;

TempStudent: PStudent;

IsCorrect: Boolean;

Begin

// Изменение базового расширения файла

OpenTextFileDialog.DefaultExt := 'grplst';

// Диалоговое окно для открытия файла

If OpenTextFileDialog.Execute Then

Begin

AssignFile(InputGroupFile, OpenTextFileDialog.FileName);

// Инициализация списка групп

TempGroupList := Nil;

// Чтение данных о группах из файла

IsCorrect := ReadFileData(InputGroupFile, InputStudentFile, TempGroupList, TempStudent, True);

// Если успешное чтение

If IsCorrect Then

Begin

// Очищение старых данных о группах

ClearGroups(GroupList);

End

Else

Begin

// Очищение новых данных

ClearGroups(TempGroupList);

// Сообщение об ошибке

Application.MessageBox('Содержимое файла повреждено!', 'Ошибка', MB\_OK + MB\_ICONERROR);

End;

End;

// Если успешное чтение данных о группах

If IsCorrect Then

Begin

// Изменение базового расширения файла

OpenTextFileDialog.DefaultExt := 'stdlst';

// Диалоговое окно открытия файла

If OpenTextFileDialog.Execute Then

Begin

// Инициализаця списка студентов

TempStudent := Nil;

AssignFile(InputStudentFile, OpenTextFileDialog.FileName);

// Чтение данных о студентах из файла

IsCorrect := ReadFileData(InputGroupFile, InputStudentFile, TempGroupList, TempStudent, False);

// Если успешное чтение

If IsCorrect Then

Begin

// Очищение старых данных студентов

ClearStudents(StudentsList);

// Запись новых данных

RecordFileData(GroupList, TempGroupList, StudentsList, TempStudent, GroupGrid);

// Изменение параметров сохранения

IsSaved := True;

SaveFile.Enabled := True;

End

Else

Begin

// Очищение новых данных студентов

ClearStudents(TempStudent);

// Сообщение об ошибке

Application.MessageBox('Содержимое файла повреждено!', 'Ошибка', MB\_OK + MB\_ICONERROR);

End;

End;

End;

End;

# ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

Основной проблемой было некорректное формирование списка студентов. При формировании возникала ошибка неверного доступа к памяти. Это было связано с тем, что не инициализировалось поле, указывающее на следующий элемент списка. И при формировании списка поле, указывающее на следующий элемент, содержало некорректные данные. Проблема представлена на рисунке 4.1.

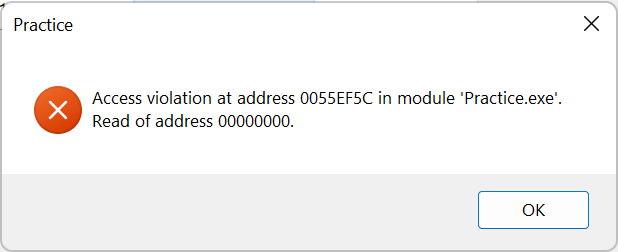


Рисунок 4.1 – Некорректное формирование списка студентов

Проблема была решена изменением функции создания нового узла, которая имеет вид:

Function CreateStudent(Const StudentData: TStudentData): PStudent;

Var

NewStudent: PStudent;

Begin

New(NewStudent);

NewStudent^.Data := StudentData;

NewStudent^.Next := Nil;

CreateStudent := NewStudent;

End;

Большинство проблем возникло из-за недочетов на стадии проектирования программного средства, на стадии тестирования приложения все проблемы были исправлены.

# РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

**5.1 Интерфейс программного средства**

Внешний вид программных средств играет существенную роль в обеспечении удобства пользователей и качества самого средства.

**5.1.1** Главное окно

Главное меню приложения состоит из четырех основных и двух дополнительных кнопок:

* Первой располагается кнопка “Список групп”, по нажатию которой пользователь переходит в модуль ViewGroupListUnit, в котором он может просмотреть список существующих групп, добавить новых, удалить или редактировать старых. Также в списке существующих групп пользователь может добавлять студентов в группы.
* Вторая кнопка “Поиск студента”, нажав на которую, пользователь может ввести ФИО студента и найти информацию об этом студенте. Данная кнопка становиться активной после того, как в списке студентов будет добавлена как минимум одна запись
* Третья кнопка “Список студентов-задолжников”, позволяет просмотреть список студентов, у которых есть неудовлетворительные оценки по дисциплинам, и просмотреть их сведения. Данная кнопка становится активной после того, как в списке студентов будет добавлена как минимум одна запись.
* Четвертая кнопка “Список студентов на отчисление” после нажатия отобразит список студентов, у которых есть 3 и более неудовлетворительных оценки по дисциплинам, и отобразит их сведения. Кнопка становится активной после того, как в списке студентов будет добавлена как минимум одна запись.
* Пятая кнопка выводит справку о программном средстве.
* Шестой кнопкой располагается кнопка для получения информации о разработчике программного средства.

Внешний вид главного меню приложения представлен на рисунке 5.1.

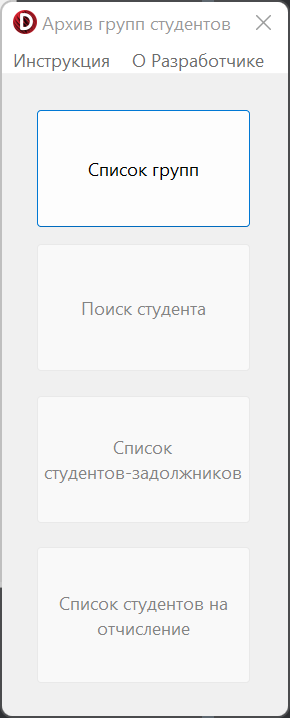
****

Рисунок 5.1 – Главное меню приложения

**5.1.2** Окно просмотра списка групп

В верхней части окна находится строка меню, где пользователь может загрузить в таблицу данные с файла или сохранить данные в файл. Данная кнопка становится неактивной, если в таблице нет записей. Ниже располагается строка с кнопками для редактирования, удаления, добавления групп, а также просмотра состава группы. Внешний вид окна просмотра списка представлен на рисунке 5.2

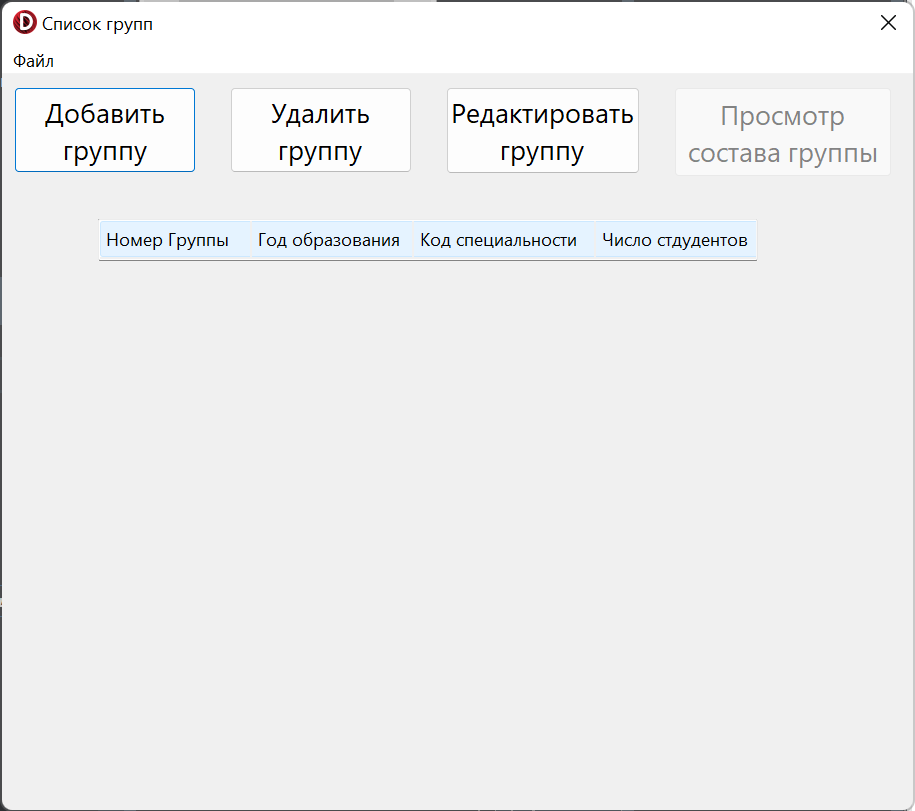


Рисунок 5.2 – Окно просмотра списка групп

**5.1.3** Окно добавления группы

Окно добавления группы состоит из текстовых полей, где пользователь должен добавить информацию о группе. Полная информация должна быть уникальной. Кнопка “Добавить” станет активной после корректного заполнения всех полей. Внешний вид окна представлен на рисунке 5.3.

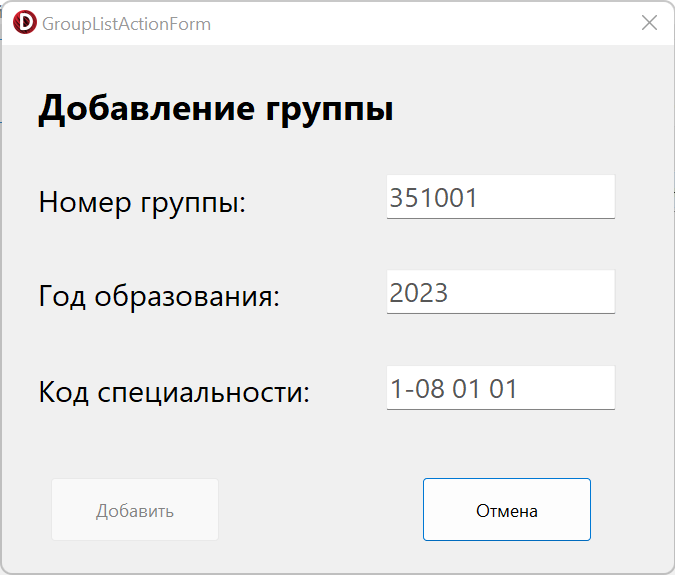


Рисунок 5.3 – Окно добавления группы

**5.1.4** Окно просмотра списка студентов в группе

В верхней части окна располагается строка с кнопками для редактирования, удаления, добавления студентов в группе. Внешний вид окна просмотра списка представлен на рисунке 5.4

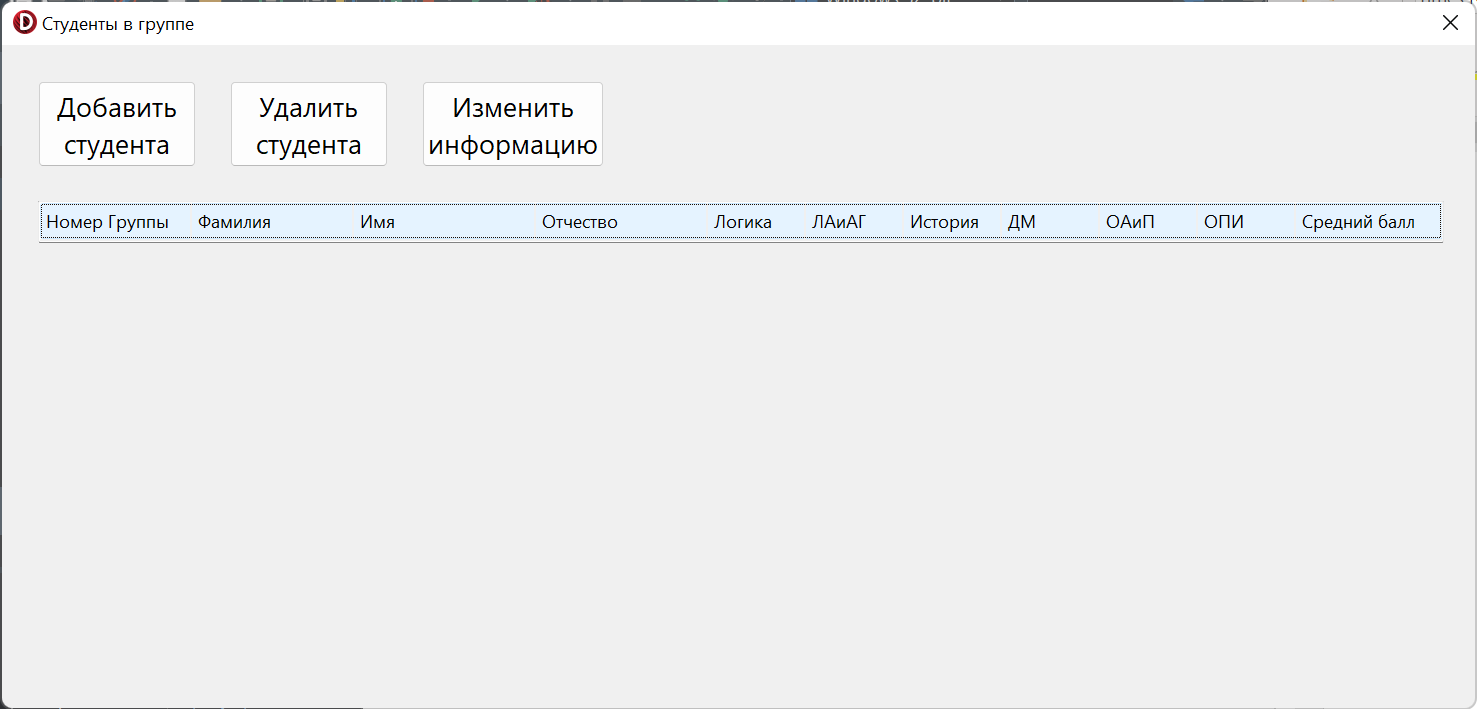


Рисунок 5.4 – Окно просмотра списка группы

**5.1.5** Окно добавления студента

Окно добавления студента состоит из текстовых полей и ячеек, где пользователь должен добавить информацию и оценки студента. Полная информация должна быть уникальной. Кнопка “Добавить” станет активной после корректного заполнения всех полей. Внешний вид окна представлен на рисунке 5.5.

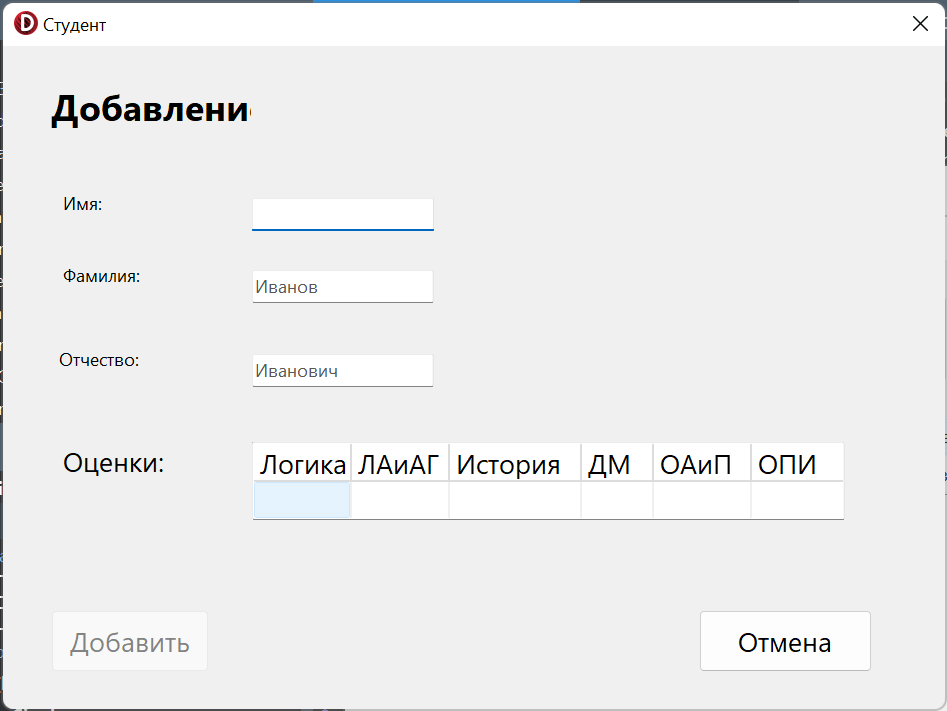
****

Рисунок 5.5 – Окно добавления студента

**5.1.6** Окно списка студентов-задолжников

Окно списка студентов-задолжников состоит из выпадающего списка и таблицы. Для получения полной информации о студентах-задолжниках вам следует щелкнуть на соответствующую дисциплину в выпадающем списке. Внешний вид окна представлен на рисунке 5.6.

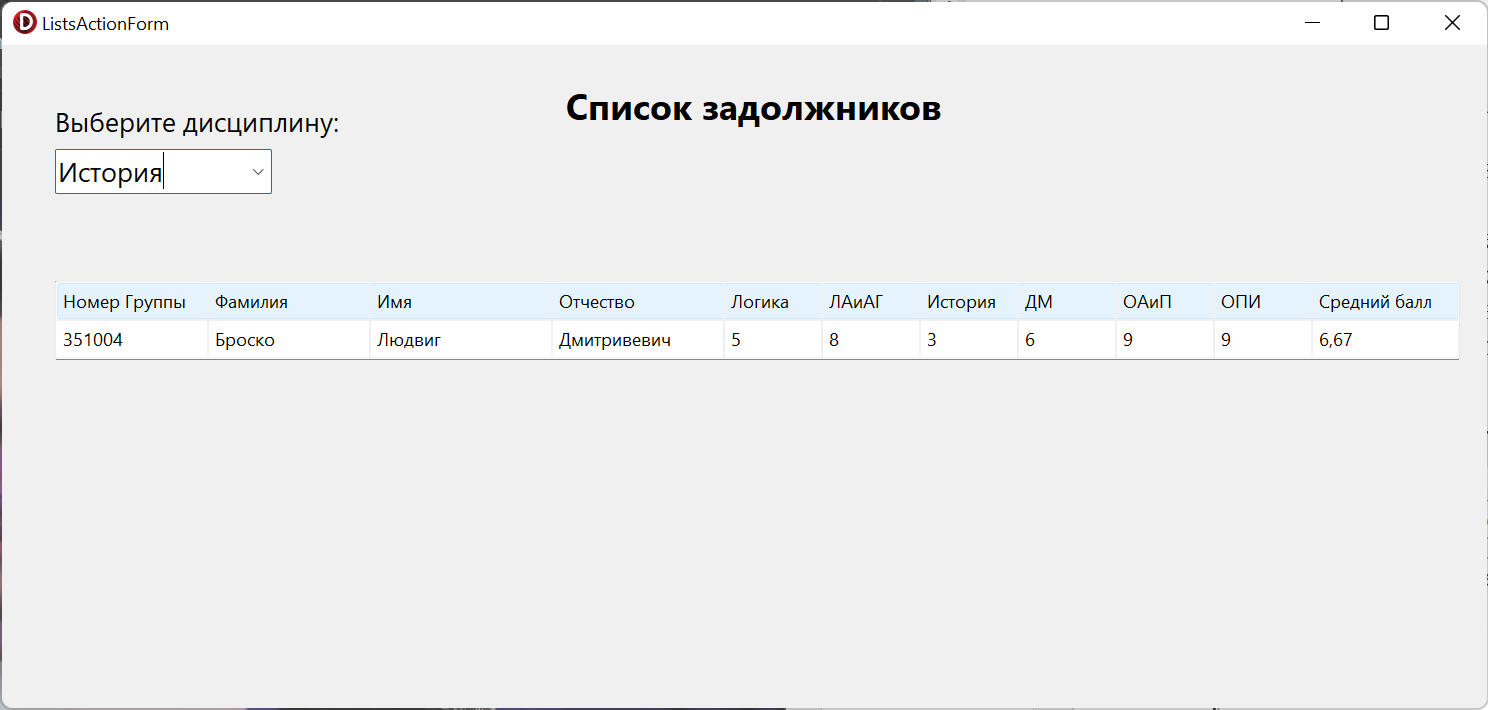


Рисунок 5.6 – Окно списка студентов-задолжников

**5.1.7** Окно списка студентов на отчисление

Окно списка студентов на отчисление, предназначено для отображения списка студентов на отчисление. Внешний вид окна списка студентов на отчисление представлен на рисунке 5.7.

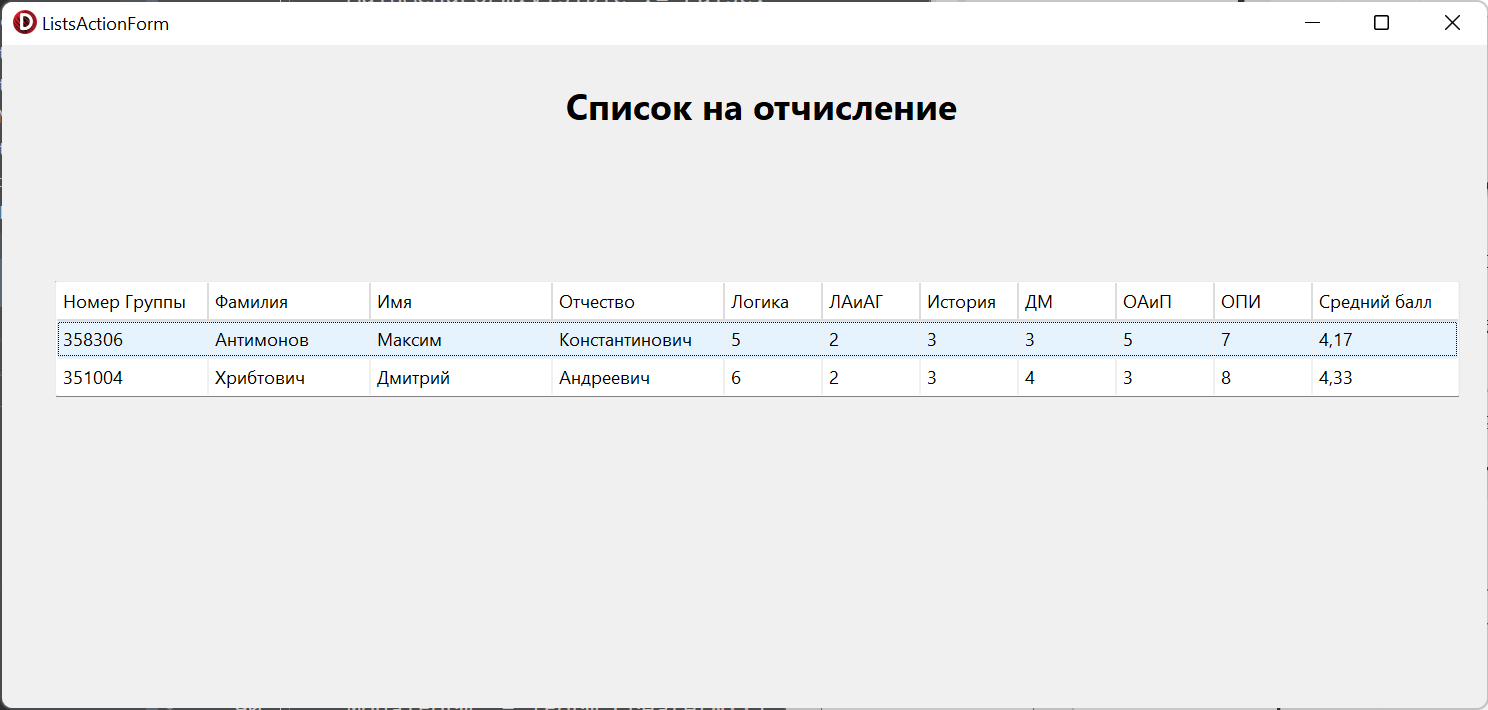


Рисунок 5.7 – Окно списка студентов на отчисление

**5.1.8** Окно поиска студента

Окно поиска студента по ФИО, предназначено для поиска студента по ФИО. Имеется кнопка “Найти” которая становится активной после проверки на наличие студента. Внешний вид окна поиска студента по ФИО представлен на рисунке 5.8.

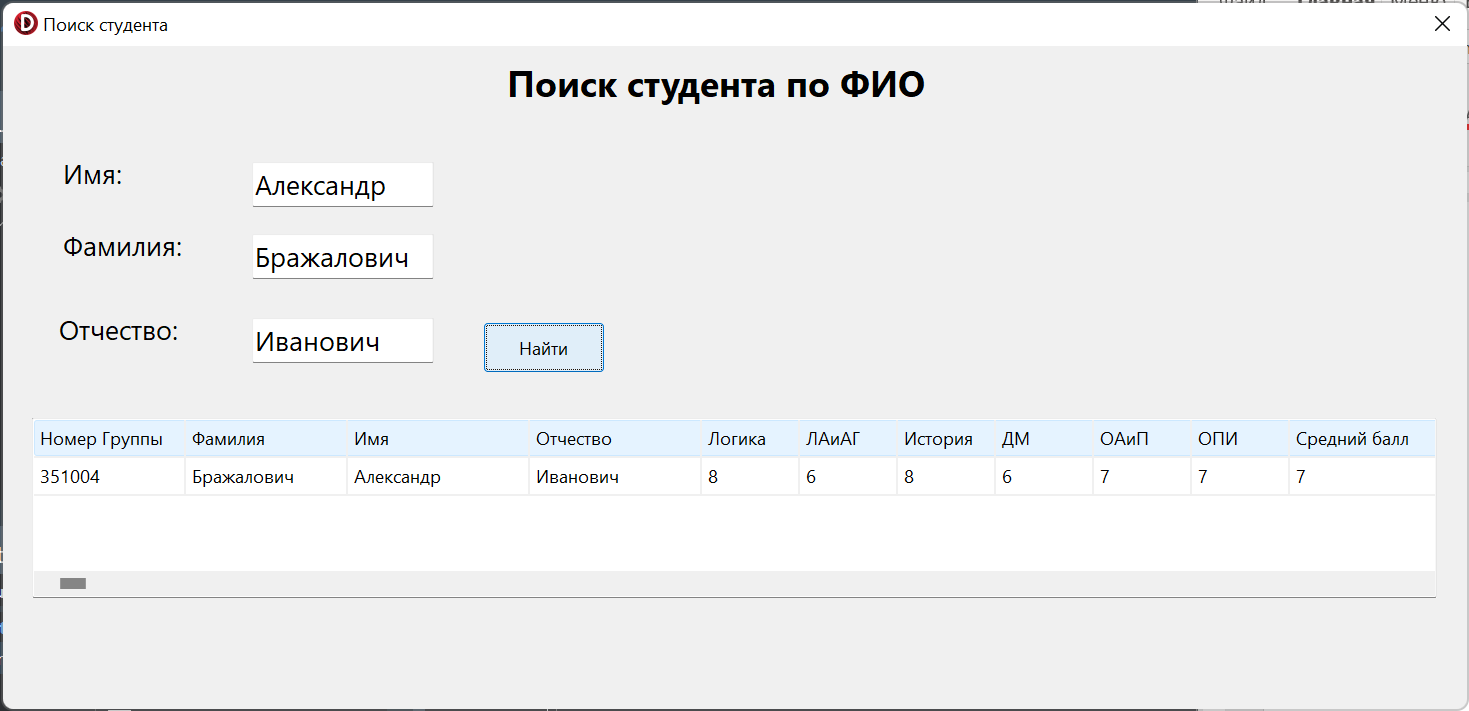


Рисунок 5.8 – Окно поиска студентов по ФИО

**5.2 Управление программным средством**

Для начала работы нужно добавить как минимум одну групу, нажав на кнопку “Список групп”. После добавления записи в список групп активной станет кнопка “Просмотр состава группы”.

Для добавления студента нужно выбрать группу из списка и нажать на кнопку “ Просмотр состава группы”. После добавления первого студента станут активными кнопки “Поиск студента”, “Список студентов-задолжников”, “Список студентов на отичсление”.

Для ввода персональных данных в программу используется два способа: ввод данных через пользовательский интерфейс, импорт данных из файла.

Ввод данных через пользовательский интерфейс осуществляется в специальном окне, где пользователь вносит данные группы в отведенные текстовые поля.

Выходными данными является полученная во время выполнения программы информация о группах, а также список студентов. Внешний вид окна списка с таблицами для редактрования представлен на рисунке 5.9.

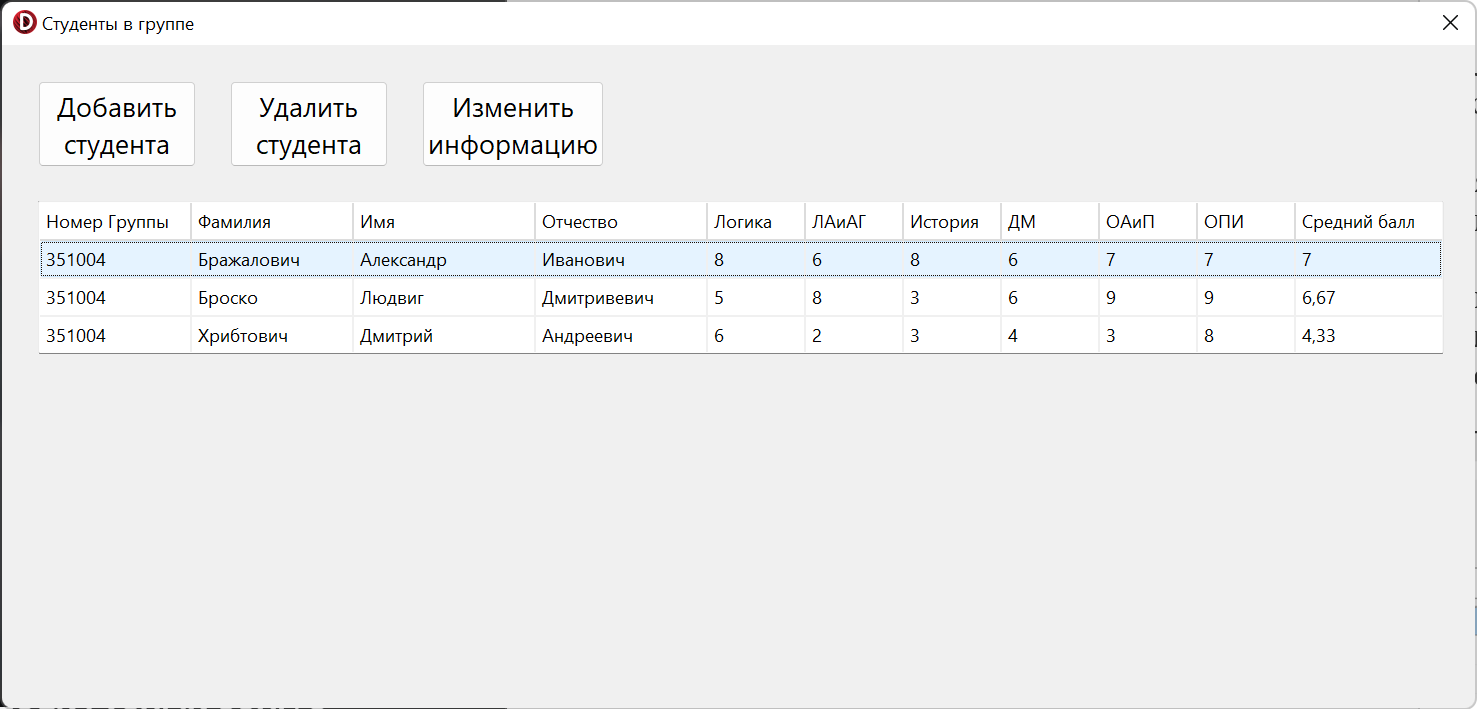


Рисунок 5.9 – Внешний вид окна списка

Вывод данных также осуществляется в двух форматах: в виде типизированного файла в конкретном формате и списками, отображающимися в виде таблиц для прямого редактирования.

**ЗАКЛЮЧЕНИЕ**

В настоящее время программные средства, позволяющие взаимодействовать со списками студентов, пользуются большой популярностью. В условиях современного образовательного процесса, где эффективное распределение времени играет важную роль, такие инструменты становятся неотъемлемой частью рабочего процесса. Высшие учебные заведения стремятся использовать современные программные решения, которые упрощают и автоматизируют управление, улучшают коммуникацию и способствуют достижению поставленных целей.

В рамках данной учебной практики было разработано программное средство для хранения и взаимодействия со списками студентов. Программное средство позволяет пользователю создавать и управлять списком групп, формировать списки студентов-задолжников, студентов на отчисление и поиска студентов по ФИО. При разработке программы были успешно выполнены все поставленные задачи, а именно:

* формирования списка студентов-задолжников;
* формирования списка студентов на отчисление;
* поиска студента и информации о нем по ФИО;
* подсистема добавления, удаления и корректирования списков;
* подсистема отображения списка целиком.

Для успешного достижения всех поставленных целей при разработке данного приложения было необходимо углубленно изучить объектно-ориентированные и формовые возможности языка программирования Delphi. Это позволило эффективно использовать функциональность данного языка при создании пользовательского интерфейса, обеспечить визуальное представление данных и взаимодействие с пользователем. Усвоение указанных аспектов языка позволило успешно реализовать все поставленные задачи в рамках разработки приложения.

В дальнейшем улучшении приложения планируется включить добавление функционала поиска групп, а также улучшение графического интерфейса.

Данное приложение позволяет оптимизировать управление задачами и ресурсами, обеспечивает легкость использования и надежность, а также эффективного взаимодействия со списками студентов.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Григорьев А.Б. О чем не пишут в книгах по Delphi [Текст]. – СПб.: БХВ-Петербург, 2010. – 576 с. – ISBN 978-5-699-40703-3.
2. Парижский С.М. Delphi. Учимся на примерах /Под ред. Ю. А. Шпака [Текст]. – Киев: МК-Пресс, 2005. – 216 с. – ISBN 978-5-9775-0516-1.
3. Синицын С.В. Программирование на языке высокого уровня / [Текст]. – М.: Академия, 2010. – 400 с. – ISBN 978-5-7695-6673-8.
4. Фаронов В. Delphi. Программирование на языке высокого уровня [Текст]: Учебник для вузов. – СПб.: Питер, 2009. – 640 с. – ISBN 978-5-9775-0657-1.
5. Фаулер М. Предметно-ориентированные языки программирования / [Текст]. – М.: Вильямс, 2011. – 576 с. – ISBN 978-5-8459-1738-6.
6. Фленов М.Е. Библия Delphi [Текст]. 2-е изд., перераб. и доп. – СПб.: БХВ-Петербург, 2008. – 800 с. – ISBN 978-5-397-01304-8.
7. Чиртик А. Программирование в Delphi. Трюки и эффекты [Текст]. – СПб.: Питер, 2011. – 400 с. – ISBN 978-5-8046-0008-3.
8. Шпак Ю.А. Delphi 7 на примерах [Текст]/Под ред. Ю.С. Ковтанюка. – Киев: Юниор, 2006. – 416 с. – ISBN 966-8806-22-0.

# 

# ПРИЛОЖЕНИЕ A

Текст программы

Unit MainMenuUnit;

Interface

Uses

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

System.Classes, Vcl.Graphics,

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

FindStudentUnit, ViewGroupListUnit, StudentLinkedListUnit,

GroupsLinkedListUnit;

Type

TStateGroup = (AddG, DeleteG, EditG);

TStateStudent = (AddS, DeleteS, EditS);

TStateLists = (Debt, Expel);

TMainMenuForm = Class(TForm)

GroupListButton: TButton;

FindStudentButton: TButton;

DebtStudentButton: TButton;

ExpellStudentButton: TButton;

DevAndInstMainMenu: TMainMenu;

InstructionMenu: TMenuItem;

DeveloperMenu: TMenuItem;

Procedure DeveloperMenuClick(Sender: TObject);

Function FormHelp(Command: Word; Data: NativeInt;Var CallHelp: Boolean): Boolean;

Procedure InstructionMenuClick(Sender: TObject);

Procedure GroupListButtonClick(Sender: TObject);

Procedure CreateForm(FormClass: TFormClass);

Procedure FindStudentButtonClick(Sender: TObject);

Procedure FormShow(Sender: TObject);

Procedure DebtStudentButtonClick(Sender: TObject);

Procedure ExpellStudentButtonClick(Sender: TObject);

Procedure FormCloseQuery(Sender: TObject; Var CanClose: Boolean);

Private

{ Private declarations }

Public

{ Public declarations }

End;

Const

MAX\_AMOUNT\_GROUPS = 50;

Var

MainMenuForm: TMainMenuForm;

StateOfGForm: TStateGroup;

StateOfSForm: TStateStudent;

GroupList: PGroup;

StudentsList: PStudent;

IsEdited: Boolean = False;

SelectedGroup: Integer = 0;

StateListForm: TStateLists;

IsSaved: Boolean = True;

Implementation

Uses

ListsActionUnit;

{$R \*.dfm}

Procedure TMainMenuForm.CreateForm(FormClass: TFormClass);

Var

Form: TForm;

Begin

MainMenuForm.Visible := False;

Form := FormClass.Create(MainMenuForm);

Form.Icon := MainMenuForm.Icon;

Form.ShowModal;

MainMenuForm.Visible := True;

End;

Procedure CreateActionForm(Const TitleLabelCaption: String);

Begin

ListsActionForm := TListsActionForm.Create(ListsActionForm);

ListsActionForm.Icon := GroupListForm.Icon;

ListsActionForm.TitleLabel.Width := 600;

ListsActionForm.TitleLabel.Caption := TitleLabelCaption;

ListsActionForm.ShowModal;

End;

Procedure CreateModalForm(CaptionText, LabelText: String;

ModalWidth, ModalHeight: Integer);

Var

ModalForm: TForm;

ModalLabel: TLabel;

Begin

ModalForm := TForm.Create(Nil);

Try

ModalForm.BorderIcons := [BiSystemMenu];

ModalForm.BorderStyle := BsSingle;

ModalForm.Caption := CaptionText;

ModalForm.Height := ModalHeight;

ModalForm.Icon := MainMenuForm.Icon;

ModalForm.Position := PoScreenCenter;

ModalForm.Width := ModalWidth;

ModalForm.OnHelp := MainMenuForm.FormHelp;

ModalLabel := TLabel.Create(ModalForm);

ModalLabel.Caption := LabelText;

ModalLabel.Font.Size := 12;

ModalLabel.Left := (ModalForm.ClientWidth - ModalLabel.Width) Div 2;

ModalLabel.Parent := ModalForm;

ModalLabel.Top := (ModalForm.ClientHeight - ModalLabel.Height) Div 2;

ModalForm.ShowModal;

Finally

ModalForm.Free;

End;

End;

Procedure TMainMenuForm.DebtStudentButtonClick(Sender: TObject);

Var

NewStudentList: PStudent;

Begin

StateListForm := Debt;

CreateActionForm('Список задолжников')

End;

Procedure TMainMenuForm.DeveloperMenuClick(Sender: TObject);

Begin

CreateModalForm('О разработчике', 'Группа: 351004'#13#10 +

'Разработчик: Бражалович Александр Иванович', 600, 150);

End;

Procedure TMainMenuForm.ExpellStudentButtonClick(Sender: TObject);

Var

NewStudentList: PStudent;

Begin

StateListForm := Expel;

NewStudentList := FindExpellStudents(StudentsList);

If NewStudentList <> Nil Then

CreateActionForm('Список на отчисление')

Else

Application.MessageBox('Студентов на отчилсение нет :(', 'Информация',

MB\_OK + MB\_ICONINFORMATION);

End;

Procedure TMainMenuForm.FindStudentButtonClick(Sender: TObject);

Begin

CreateForm(TFindStudentForm);

End;

Procedure TMainMenuForm.FormCloseQuery(Sender: TObject; Var CanClose: Boolean);

Var

Confirmation: Integer;

Begin

If IsSaved Then

Begin

Confirmation := Application.MessageBox('Вы действительно хотите выйти?',

'Выход', MB\_YESNO + MB\_ICONQUESTION + MB\_DEFBUTTON2);

CanClose := Confirmation = IDYES;

End

Else

Begin

Confirmation := Application.MessageBox

('Вы не сохранили файл, хотите ли сохранить файл?', 'Выход',

MB\_YESNOCANCEl + MB\_ICONQUESTION + MB\_DEFBUTTON2);

Case Confirmation Of

MrYes:

Begin

GroupListForm.SaveFileClick(Sender);

CanClose := IsSaved;

End;

MrNo:

CanClose := True;

MrCancel:

CanClose := False;

End;

End;

End;

Function TMainMenuForm.FormHelp(Command: Word; Data: NativeInt;

Var CallHelp: Boolean): Boolean;

Begin

CallHelp := False;

FormHelp := False;

End;

Procedure TMainMenuForm.FormShow(Sender: TObject);

Begin

FindStudentButton.Enabled := StudentsList <> Nil;

ExpellStudentButton.Enabled := StudentsList <> Nil;

DebtStudentButton.Enabled := StudentsList <> Nil;

End;

Procedure TMainMenuForm.GroupListButtonClick(Sender: TObject);

Begin

CreateForm(TGroupListForm);

End;

Procedure TMainMenuForm.InstructionMenuClick(Sender: TObject);

Begin

CreateModalForm('Инструкция', 'Список групп:'#13#10 +

'1. Для добавления кандидата нажмите кнопку "Добавить".'#13#10

+ '2. Для изменения группы нажмите кнопку "Изменить".'#13#10

+ '3. Для удаления группы нажмите кнопку "Удалить".'#13#10

+ '4. Для получения диапазона для ввода поля наведитесь на него.'#13#10#13#10

+ 'Файлы:'#13#10 + '1. Файл имеет расширение .grplst'#13#10#13#10 +

'Поиск:'#13#10 +

'Для использования поиска введите ФИО студента и нажмите кнопку "Найти".'#13#10#13#10

+ 'Работает на русской системе Windows.', 1000, 450);

End;

End.

Unit GroupListActionUnit;

Interface

Uses

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

System.Classes, Vcl.Graphics,

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

Type

TGroupListActionForm = Class(TForm)

TitleLabel: TLabel;

ActionButton: TButton;

CancelButton: TButton;

NumberLabel: TLabel;

YearLabel: TLabel;

CodeLabel: TLabel;

NumberEdit: TEdit;

YearEdit: TEdit;

CodeEdit: TEdit;

Procedure CreateParams(Var Params: TCreateParams); Override;

Procedure CancelButtonClick(Sender: TObject);

Procedure EditContextPopup(Sender: TObject; MousePos: TPoint; Var Handled: Boolean);

Procedure NumberEditKeyPress(Sender: TObject; Var Key: Char);

Procedure YearEditKeyPress(Sender: TObject; Var Key: Char);

Procedure CodeEditKeyPress(Sender: TObject; Var Key: Char);

Procedure EditChange(Sender: TObject);

Procedure ActionButtonClick(Sender: TObject);

Procedure EditKeyDown(Sender: TObject; Var Key: Word;

Shift: TShiftState);

Procedure FormClose(Sender: TObject; Var Action: TCloseAction);

Procedure FormShow(Sender: TObject);

Private

{ Private declarations }

Public

{ Public declarations }

End;

Var

GroupListActionForm: TGroupListActionForm;

Implementation

Uses

InputEditUnit, GroupsLinkedListUnit, MainMenuUnit, ViewGroupListUnit;

{$R \*.dfm}

Procedure TGroupListActionForm.CreateParams(Var Params: TCreateParams);

Begin

Inherited;

Params.ExStyle := Params.ExStyle Or WS\_EX\_APPWINDOW;

End;

Function CreateGroup(): TGroupData;

Var

NewGroup: TGroupData;

Begin

With NewGroup, GroupListActionForm Do

Begin

GroupNumber := StrToInt(NumberEdit.Text);

Year := StrToInt(YearEdit.Text);

Code := CodeEdit.Text;

CountOfStudents := 0;

End;

CreateGroup := NewGroup;

End;

Procedure TGroupListActionForm.ActionButtonClick(Sender: TObject);

Var

NewGroupData: TGroupData;

Begin

NewGroupData := CreateGroup();

If StateOfGForm = AddG Then

Begin

AddGroup(GroupList, NewGroupData);

End

Else If StateOfGForm = EditG Then

Begin

EditGroup(GroupList, OldGroupData, NewGroupData, StudentsList);

End;

IsEdited := True;

Close;

End;

Procedure TGroupListActionForm.CancelButtonClick(Sender: TObject);

Begin

IsEdited := False;

Close;

End;

Procedure TGroupListActionForm.EditContextPopup(Sender: TObject;

MousePos: TPoint; Var Handled: Boolean);

Begin

Handled := True;

End;

Procedure TGroupListActionForm.CodeEditKeyPress(Sender: TObject; Var Key: Char);

Begin

CheckComboButtons(Key, CodeEdit, NUMBERS\_EXT);

CheckLength(CODE\_LENGTH, CodeEdit, Key);

End;

Procedure TGroupListActionForm.EditChange(Sender: TObject);

Var

NewGroupData: TGroupData;

Begin

ActionButton.Enabled := False;

If IsFullFields(NumberEdit, YearEdit, CodeEdit) Then

Begin

NewGroupData := CreateGroup();

ActionButton.Enabled := Not IsExistGroup(GroupList, NewGroupData);

End;

End;

Procedure TGroupListActionForm.EditKeyDown(Sender: TObject; Var Key: Word;

Shift: TShiftState);

Begin

CheckShiftAndArrows(Key, Shift);

End;

Procedure TGroupListActionForm.FormClose(Sender: TObject;

Var Action: TCloseAction);

Begin

IsEdited := False

End;

Procedure SetGroupData(OldGroupData: TGroupData);

Begin

With GroupListActionForm, OldGroupData Do

Begin

NumberEdit.Text := IntToStr(GroupNumber);

YearEdit.Text := IntToStr(Year);

CodeEdit.Text := String(Code);

End;

End;

Procedure TGroupListActionForm.FormShow(Sender: TObject);

Begin

If StateOfGForm = EditG Then

SetGroupData(OldGroupData);

End;

Procedure TGroupListActionForm.NumberEditKeyPress(Sender: TObject;

Var Key: Char);

Begin

CheckLength(GROUP\_LENGTH, NumberEdit, Key);

CheckComboButtons(Key, NumberEdit, NUMBERS);

End;

Procedure TGroupListActionForm.YearEditKeyPress(Sender: TObject; Var Key: Char);

Begin

CheckComboButtons(Key, YearEdit, NUMBERS);

CheckNumberRange(Key, YEAR\_MAX, YEAR\_MIN, YearEdit.Text)

End;

End.

Unit GroupsLinkedListUnit;

Interface

Uses

GroupListActionUnit, InputEditUnit, StudentLinkedListUnit, VCL.Grids,

SysUtils, VCL.StdCtrls;

Type

TGroupNumber = Integer;

TYear = Integer;

TCodeString = String[CODE\_LENGTH];

TGroupData = Record

GroupNumber: TGroupNumber;

Year: TYear;

Code: TCodeString;

CountOfStudents: Integer;

End;

PGroup = ^TGroup;

TGroup = Record

Data: TGroupData;

Next: PGroup;

End;

Const

MAX\_COUNT\_OF\_STUDENTS = 40;

MAX\_COUNT\_OF\_GROUPS = 50;

Function IsExistGroup(Const GroupList: PGroup;

Const GroupData: TGroupData): Boolean;

Procedure AddGroup(Var GroupList: PGroup; Const GroupData: TGroupData);

Function CreateGroup(Const GroupData: TGroupData): PGroup;

Function IsGroupDataEqual(Const GroupData1, GroupData2: TGroupData): Boolean;

Procedure DrawGroupGrid(Var GroupGrid: TStringGrid; GroupList: PGroup);

Function CountGroups(Const GroupList: PGroup): Integer;

Function FindCountInGroup(Const GroupList: PGroup;

CurrGroupNumber: Integer): Integer;

Procedure AddStudentToGroup(Var GroupList: PGroup; CurrGroupNumber: Integer);

Procedure DeleteGroupF(Var GroupList: PGroup; Const GroupData: TGroupData;

Var StudentList: PStudent);

Procedure DeleteOneStudent(Var GroupList: PGroup; CurrGroupNumber: Integer);

Procedure EditGroup(Const GroupList: PGroup;

Const OldGroupData, NewGroupData: TGroupData; Var StudentList: PStudent);

Function IsExistGroupByNumber(Const GroupList: PGroup;

CurrGroupNumber: Integer): Boolean;

Procedure EditStudent(Var GroupList: PGroup;

Const OldStudentData, NewStudentData: TStudentData;

Var StudentsList: PStudent);

Procedure ClearGroups(Var GroupList: PGroup);

Implementation

Function SearchGroup(Const GroupList: PGroup;

Const GroupData: TgroupData): PGroup;

Var

CurrGroup: PGroup;

Begin

CurrGroup := GroupList;

While Not IsGroupDataEqual(CurrGroup^.Data, GroupData) Do

CurrGroup := CurrGroup^.Next;

SearchGroup := CurrGroup;

End;

Procedure EditGroup(Const GroupList: PGroup;

Const OldGroupData, NewGroupData: TGroupData; Var StudentList: PStudent);

Var

CurrGroup: PGroup;

CurrStudent: PStudent;

PrevGroupNumber: Integer;

PrevCountOfStudents: Integer;

NewGroupNumber: Integer;

Begin

PrevGroupNumber := OldGroupData.GroupNumber;

PrevCountOfStudents := OldGroupData.CountOfStudents;

CurrGroup := SearchGroup(GroupList, OldGroupData);

CurrGroup^.Data := NewGroupData;

CurrGroup^.Data.CountOfStudents := PrevCountOfStudents;

NewGroupNumber := NewGroupData.GroupNumber;

CurrStudent := StudentList;

While CurrStudent <> Nil Do

Begin

If (CurrStudent^.Data.GroupNumber = PrevGroupNumber) Then

Begin

CurrStudent^.Data.GroupNumber := NewGroupNumber;

End;

CurrStudent := CurrStudent^.Next;

End;

End;

Function IsExistGroupByNumber(Const GroupList: PGroup;

CurrGroupNumber: Integer): Boolean;

Var

CurrGroup: PGroup;

IsExist: Boolean;

Begin

CurrGroup := GroupList;

IsExist := False;

If CurrGroup <> Nil Then

Repeat

IsExist := CurrGroup^.Data.GroupNumber = CurrGroupNumber;

CurrGroup := CurrGroup^.Next

Until (IsExist) Or (CurrGroup = Nil);

IsExistGroupByNumber := IsExist;

End;

Function SearchStudent(Const StudentList: PStudent;

Const StudentData: TStudentData): PStudent;

Var

CurrStudent: PStudent;

Begin

CurrStudent := StudentList;

While Not IsStudentDataEqual(CurrStudent^.Data, StudentData) And

(CurrStudent <> Nil) Do

CurrStudent := CurrStudent^.Next;

SearchStudent := CurrStudent;

End;

Function SearchGroupByNumber(Const GroupList: PGroup; CurrGroupNumber: Integer)

: TGroupData;

Var

CurrGroup: PGroup;

IsExist: Boolean;

Begin

CurrGroup := GroupList;

IsExist := False;

If CurrGroup <> Nil Then

Repeat

IsExist := CurrGroup^.Data.GroupNumber = CurrGroupNumber;

CurrGroup := CurrGroup^.Next

Until (IsExist) Or (CurrGroup = Nil);

SearchGroupByNumber := CurrGroup^.Data;

End;

Procedure EditStudent(Var GroupList: PGroup;

Const OldStudentData, NewStudentData: TStudentData;

Var StudentsList: PStudent);

Var

PrevGroupNumber: Integer;

PrevCountOfStudents: Integer;

NewGroupNumber: Integer;

CurrGroup: PGroup;

CurrStudent: PStudent;

NewGroupData: TGroupdata;

Begin

PrevGroupNumber := OldStudentData.GroupNumber;

NewGroupNumber := NewStudentData.GroupNumber;

CurrStudent := SearchStudent(StudentsList, OldStudentData);

AddStudent(StudentsList, NewStudentData);

AddStudentToGroup(GroupList, NewGroupNumber);

DeleteOneStudent(GroupList, PrevGroupNumber);

DeleteStudent(StudentsList, OldStudentData);

End;

Procedure DeleteOneStudent(Var GroupList: PGroup; CurrGroupNumber: Integer);

Var

CurrGroup: PGroup;

NewCount: Integer;

Begin

CurrGroup := GroupList;

While (CurrGroup <> Nil) And

(CurrGroup^.Data.GroupNumber <> CurrGroupNumber) Do

Begin

CurrGroup := CurrGroup^.Next;

End;

Dec(CurrGroup^.Data.CountOfStudents);

End;

Procedure ClearGroups(Var GroupList: PGroup);

Var

CurrGroup, TempGroup: PGroup;

Begin

CurrGroup := GroupList;

While CurrGroup <> Nil Do

Begin

TempGroup := CurrGroup;

CurrGroup := CurrGroup^.Next;

Dispose(TempGroup);

End;

GroupList := Nil;

End;

Procedure DeleteGroupF(Var GroupList: PGroup; Const GroupData: TGroupData;

Var StudentList: PStudent);

Var

CurrGroup, TempGroup: PGroup;

Begin

CurrGroup := GroupList;

If IsGroupDataEqual(CurrGroup^.Data, GroupData) Then

Begin

DeleteStudentsInGroup(GroupList^.Data.GroupNumber, StudentList);

GroupList := CurrGroup^.Next;

Dispose(CurrGroup);

End

Else

Begin

While Not IsGroupDataEqual(CurrGroup^.Next^.Data, GroupData) Do

CurrGroup := CurrGroup^.Next;

TempGroup := CurrGroup^.Next;

DeleteStudentsInGroup(TempGroup^.Data.GroupNumber, StudentList);

CurrGroup^.Next := CurrGroup^.Next^.Next;

Dispose(TempGroup);

End;

End;

Procedure AddStudentToGroup(Var GroupList: PGroup; CurrGroupNumber: Integer);

Var

CurrGroup: PGroup;

NewCount: Integer;

Begin

CurrGroup := GroupList;

While (CurrGroup <> Nil) And

(CurrGroup^.Data.GroupNumber <> CurrGroupNumber) Do

Begin

CurrGroup := CurrGroup^.Next;

End;

Inc(CurrGroup^.Data.CountOfStudents);

End;

Function FindCountInGroup(Const GroupList: PGroup;

CurrGroupNumber: Integer): Integer;

Var

CurrGroup: PGroup;

Count: Integer;

Begin

CurrGroup := GroupList;

Count := -1;

While (CurrGroup <> Nil) And (Count = -1) Do

Begin

If (CurrGroup^.Data.GroupNumber = CurrGroupNumber) Then

Count := CurrGroup^.Data.CountOfStudents;

CurrGroup := CurrGroup^.Next;

End;

FindCountInGroup := Count;

End;

Function CreateGroup(Const GroupData: TGroupData): PGroup;

Var

NewGroup: PGroup;

Begin

New(NewGroup);

NewGroup^.Data := GroupData;

NewGroup^.Next := Nil;

CreateGroup := NewGroup;

End;

Procedure AddGroup(Var GroupList: PGroup; Const GroupData: TGroupData);

Var

NewGroup, CurrGroup: PGroup;

Begin

NewGroup := CreateGroup(GroupData);

If GroupList = Nil Then

GroupList := NewGroup

Else

Begin

CurrGroup := GroupList;

While CurrGroup^.Next <> Nil Do

CurrGroup := CurrGroup^.Next;

CurrGroup^.Next := NewGroup;

End;

End;

Function IsGroupDataEqual(Const GroupData1, GroupData2: TGroupData): Boolean;

Begin

IsGroupDataEqual := (GroupData1.GroupNumber = GroupData2.GroupNumber)

End;

Function IsExistGroup(Const GroupList: PGroup;

Const GroupData: TGroupData): Boolean;

Var

CurrGroup: PGroup;

IsExist: Boolean;

Begin

CurrGroup := GroupList;

IsExist := False;

If CurrGroup <> Nil Then

Repeat

IsExist := IsGroupDataEqual(CurrGroup^.Data, GroupData);

CurrGroup := CurrGroup^.Next

Until (IsExist) Or (CurrGroup = Nil);

IsExistGroup := IsExist;

End;

Function CountGroups(Const GroupList: PGroup): Integer;

Var

CurrGroup: PGroup;

GroupCount: Integer;

Begin

CurrGroup := GroupList;

GroupCount := 0;

While CurrGroup <> Nil Do

Begin

Inc(GroupCount);

CurrGroup := CurrGroup^.Next;

End;

CountGroups := GroupCount;

End;

Procedure DrawGroupGrid(Var GroupGrid: TStringGrid; GroupList: PGroup);

Var

CurrGroup: PGroup;

CurrRow: Integer;

Begin

With GroupGrid Do

Begin

Width := 660;

Cells[0, 0] := 'Номер Группы';

Cells[1, 0] := 'Год образования';

Cells[2, 0] := 'Код специальности';

Cells[3, 0] := 'Число стдудентов';

ColWidths[0] := 150;

ColWidths[1] := 160;

ColWidths[2] := 180;

ColWidths[3] := 170;

End;

CurrGroup := GroupList;

CurrRow := 1;

While CurrGroup <> Nil Do

Begin

With GroupGrid, CurrGroup^.Data Do

Begin

Cells[0, CurrRow] := IntToStr(GroupNumber);

Cells[1, CurrRow] := IntToStr(Year);

Cells[2, CurrRow] := String(Code);

Cells[3, CurrRow] := IntToStr(CountOfStudents);

End;

Inc(CurrRow);

CurrGroup := CurrGroup^.Next;

End;

If GroupGrid.RowCount > 10 Then

Begin

GroupGrid.ScrollBars := TScrollStyle.SsVertical;

GroupGrid.Height := 400

End

Else

Begin

GroupGrid.ScrollBars := TScrollStyle.SsNone;

GroupGrid.Height :=

(GroupGrid.DefaultRowHeight + GroupGrid.GridLineWidth) \*

GroupGrid.RowCount + 5;

End;

End;

End.

Unit InputEditUnit;

Interface

Uses

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

System.Variants, Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls,

Vcl.Grids;

Type

TGrid = Class(TStringGrid);

Const

GROUP\_LENGTH = 6;

YEAR\_LENGTH = 4;

CODE\_LENGTH = 10;

NAME\_LENGTH = 15;

MARK\_MIN = 0;

MARK\_MAX = 10;

YEAR\_MIN = 1964;

YEAR\_MAX = 2100;

NONE = #0;

BACKSPACE = #8;

NUMBERS = ['0' .. '9'];

NUMBERS\_EXT = ['0' .. '9', '-'];

LETTERS = ' - абвгдеёжзийклмнопрстуфхцчшщъыьэюяabcdefghijklmnopqrstuvwxyzАБВГДЕЁЖЗИЙКЛМНОПРС ТУФХЦЧШЩЪЫЬЭЮЯABCDEFGHIJKLMNOPQRSTUVWXYZ';

Procedure CheckShiftAndArrows(Var Key: Word; Shift: TShiftState);

Procedure CheckComboButtons(Var Key: Char; Edit: TEdit;

Const Chars: TSysCharSet);

Procedure CheckLength(Const MAX: Integer; Edit: TEdit; Var Key: Char);

Function IsCorrectRange(Value: Integer; Const MIN, MAX: Integer): Boolean;

Procedure CheckNumberRange(Var Key: Char; Const MAX, MIN: Integer;

Text: String);

Function IsFullFields(NumberEdit, YearEdit, CodeEdit: TEdit): Boolean;

Function IsAllCellFill(Grid: TStringGrid): Boolean;

Function IsFullFieldsStudents(NameEdit, SurnameEdit, PatrynomicEdit,

GroupNumberEdit: TEdit; Grid: TStringGrid): Boolean;

Procedure CheckValidMarks(Var MarksGrid: TStringGrid; Var Key: Char);

Function IsValidChars(Const Text: String): Boolean;

Function IsValidChar(Const Key: Char): Boolean;

Function IsValidRange(Const Num, MAX: Integer): Boolean;

Procedure NameComponentKeyPress(Const Text: String;

Const SelStart, SelLength, MAX: Integer; Var Key: Char);

Function IsFullFieldsStudentsNSP(NameEdit, SurnameEdit, PatrynomicEdit

: TEdit): Boolean;

Implementation

Procedure NameComponentKeyPress(Const Text: String;

Const SelStart, SelLength, MAX: Integer; Var Key: Char);

Begin

If (Key <> BACKSPACE) Then

Begin

If Not(IsValidChar(Key) And

IsValidRange(Length(Copy(Text, 1, SelStart) + Key + Copy(Text,

SelStart + SelLength + 1)), MAX)) Then

Key := NONE;

End;

End;

Function IsValidRange(Const Num, MAX: Integer): Boolean;

Begin

IsValidRange := Num <= MAX;

End;

Procedure CheckShiftAndArrows(Var Key: Word; Shift: TShiftState);

Begin

If (Key = VK\_INSERT) And (Shift = [SsShift]) Then

Key := Ord(NONE);

If (Key = VK\_LEFT) Or (Key = VK\_UP) Then

Key := Ord(NONE)

Else If (Key = VK\_RIGHT) Or (Key = VK\_DOWN) Then

Key := Ord(NONE);

End;

Procedure CheckComboButtons(Var Key: Char; Edit: TEdit;

Const Chars: TSysCharSet);

Begin

If (Key = #22) Or ((Key = 'v') And (GetKeyState(VK\_CONTROL) < 0)) Then

Key := #0;

If Not CharInSet(Key, Chars) And (Key <> #8) Then

Key := #0;

End;

Procedure CheckNumberRange(Var Key: Char; Const MAX, MIN: Integer;

Text: String);

Var

Value: Integer;

Begin

Value := 0;

If TryStrToInt(Text + Key, Value) Then

Begin

If Not IsCorrectRange(Value, 0, MAX) Then

Begin

Key := #0;

End;

End;

End;

Function IsCorrectRange(Value: Integer; Const MIN, MAX: Integer): Boolean;

Begin

IsCorrectRange := Not((Value < MIN) Or (Value > MAX));

End;

Function IsValidChar(Const Key: Char): Boolean;

Var

C: Char;

Begin

For C In LETTERS Do

If C = Key Then

Exit(True);

IsValidChar := False;

End;

Function IsValidChars(Const Text: String): Boolean;

Var

I: Integer;

Begin

For I := 1 To Length(Text) Do

If Not IsValidChar(Text[I]) Then

Exit(False);

IsValidChars := True;

End;

Procedure CheckLength(Const MAX: Integer; Edit: TEdit; Var Key: Char);

Var

BufString: String;

Begin

If (Length(Edit.Text) >= MAX) And (Key <> BACKSPACE) Then

Key := NONE;

End;

Function IsFullFields(NumberEdit, YearEdit, CodeEdit: TEdit): Boolean;

Begin

IsFullFields := (Trim(NumberEdit.Text) <> '') And

(Trim(YearEdit.Text) <> '') And (Trim(CodeEdit.Text) <> '') And

(Length(Trim(NumberEdit.Text)) = GROUP\_LENGTH) And

(Length(Trim(YearEdit.Text)) = YEAR\_LENGTH) And

(Length(Trim(CodeEdit.Text)) = CODE\_LENGTH);

End;

Function IsAllCellFill(Grid: TStringGrid): Boolean;

Var

IsFilled: Boolean;

Col, Row, Num: Integer;

Begin

IsFilled := True;

Row := 0;

While IsFilled And (Row < Grid.RowCount) Do

Begin

Col := 0;

While IsFilled And (Col < Grid.ColCount) Do

Begin

If (Grid.Cells[Col, Row] = '') Then

IsFilled := False;

Inc(Col);

End;

Inc(Row);

End;

IsAllCellFill := IsFilled;

End;

Function IsValidRangeGrid(Text: String; Key: Char;

SelStart, MIN, Max: Integer): Boolean;

Var

IsValidInput: Boolean;

Num: Integer;

Str: String;

Begin

IsValidInput := True;

Str := Copy(Text, 0, SelStart) + Key + Copy(Text, SelStart + 1);

If Not TryStrToInt(Str, Num) Then

IsValidInput := False;

If (Num < MIN) Or (Num > MAX) Then

IsValidInput := False;

IsValidRangeGrid := IsValidInput;

End;

Procedure ComponentKeyPressGrid(Var Key: Char; SelStart, SelLength: Integer;

Text: String; Const MIN\_NUM, MAX\_NUM: Integer);

Var

IsValidInput: Boolean;

Begin

IsValidInput := (CharInSet(Key, NUMBERS) Or (Key = BACKSPACE));

If (SelLength > 0) And (SelLength < Length(Text)) Then

Key := NONE

Else

Begin

If (SelLength = Length(Text)) And (Key = '0') And

(Length(Text) <> 0) Then

Key := NONE;

If (SelStart = 0) And (SelLength < Length(Text)) And (Key = '0') Then

Key := NONE;

If (SelStart = 1) And (Length(Text) > 1) And (Text[2] = '0') And

(Key = BACKSPACE) Then

Key := NONE;

If (IsValidInput) And (Key <> BACKSPACE) And (Key <> NONE) Then

IsValidInput := IsValidRangeGrid(Text, Key, SelStart,

MIN\_NUM, MAX\_NUM);

If Not IsValidInput Then

Key := NONE;

End;

End;

Procedure CheckValidMarks(Var MarksGrid: TStringGrid; Var Key: Char);

Var

NumsGrid: TGrid;

SelStart, SelLength: Integer;

Text: String;

Begin

NumsGrid := TGrid(MarksGrid);

If Assigned(NumsGrid.InplaceEditor) Then

Begin

NumsGrid.InplaceEditor.Visible;

SelStart := NumsGrid.InplaceEditor.SelStart;

SelLength := NumsGrid.InplaceEditor.SelLength;

Text := NumsGrid.InplaceEditor.Text;

ComponentKeyPressGrid(Key, SelStart, SelLength, Text, 1, 10);

End;

End;

Function IsFullFieldsStudents(NameEdit, SurnameEdit, PatrynomicEdit,

GroupNumberEdit: TEdit; Grid: TStringGrid): Boolean;

Var

IsFull: Boolean;

Begin

IsFull := False;

IsFull := IsAllCellFill(Grid) And (Trim(NameEdit.Text) <> '') And

(Trim(SurnameEdit.Text) <> '') And (Trim(PatrynomicEdit.Text) <> '');

IsFullFieldsStudents := IsFull;

End;

Function IsFullFieldsStudentsNSP(NameEdit, SurnameEdit, PatrynomicEdit

: TEdit): Boolean;

Var

IsFull: Boolean;

Begin

IsFull := False;

IsFull := (Trim(NameEdit.Text) <> '') And (Trim(SurnameEdit.Text) <> '') And

(Trim(PatrynomicEdit.Text) <> '');

IsFullFieldsStudentsNSP := IsFull;

End;

Exports CheckLength, CheckComboButtons, CheckShiftAndArrows, CheckNumberRange,

IsFullFields;

End.

Unit StudentLinkedListUnit;

Interface

Uses

Vcl.Grids, Vcl.StdCtrls, SysUtils;

Const

NAME\_LENGTH = 15;

Type

Diciplines = (Logic, Algebra, History, DM, BAAP, BPE);

TNameString = String[NAME\_LENGTH];

TDicipline = Array [Diciplines] Of Integer;

TGroupNumber = Integer;

TStudentData = Record

GroupNumber: TGroupNumber;

Surname: TNameString;

Name: TNameString;

Patronymic: TNameString;

Marks: TDicipline;

AvgMark: Real;

End;

PStudent = ^TStudent;

TStudent = Record

Data: TStudentData;

Next: PStudent;

End;

Procedure DeleteStudentsInGroup(GroupNumber: Integer;

Var StudentList: PStudent);

Procedure DrawMarksGrid(Var MarkGrid: TStringGrid);

Procedure AddStudent(Var StudentList: PStudent;

Const StudentData: TStudentData);

Function IsExistStudent(Const StudentList: PStudent;

Const StudentData: TStudentData): Boolean;

Procedure DrawStudentsInGroupGrid(Var StudentsInGroupGrid: TStringGrid;

StudentsList: PStudent; CurrentGroup: Integer);

Function IsStudentDataEqual(Const StudentData1, StudentData2

: TStudentData): Boolean;

Procedure DeleteStudent(Var StudentList: PStudent;

Const StudentData: TStudentData);

Function IsExistStudentNSP(Const StudentList: PStudent;

Name, Surname, Patronymic: TNameString): Boolean;

Function SearchStudentNSP(StudentList: PStudent;

Name, Surname, Patronymic: TNameString): PStudent;

Function FindExpellStudents(Const StudentsList: PStudent): PStudent;

Function FindDeptStudents(StudentsList: PStudent; CurrDicipline: Integer)

: PStudent;

Procedure ClearStudentGrid(Var StudentsGrid: TStringGrid);

Procedure ClearStudents(Var StudentList: PStudent);

Implementation

Uses

MainMenuUnit, GroupsLinkedListUnit;

Function FindExpellStudents(Const StudentsList: PStudent): PStudent;

Var

CurrStudent: PStudent;

ExpellStudents: PStudent;

Count, I: Integer;

Begin

ExpellStudents := Nil;

CurrStudent := StudentsList;

Count := 0;

While CurrStudent <> Nil Do

Begin

With CurrStudent^.Data Do

Begin

If Marks[Logic] < 4 Then

Inc(Count);

If Marks[Algebra] < 4 Then

Inc(Count);

If Marks[History] < 4 Then

Inc(Count);

If Marks[BAAP] < 4 Then

Inc(Count);

If Marks[BPE] < 4 Then

Inc(Count);

If Marks[DM] < 4 Then

Inc(Count);

If Count > 2 Then

AddStudent(ExpellStudents, CurrStudent^.Data)

End;

CurrStudent := CurrStudent^.Next;

Count := 0;

End;

FindExpellStudents := ExpellStudents;

End;

Procedure ClearStudents(Var StudentList: PStudent);

Var

CurrStudent, TempStudent: Pstudent;

Begin

CurrStudent := StudentList

While CurrStudent <> Nil Do

Begin

TempStudent := CurrStudent;

CurrStudent := CurrStudent^.Next;

Dispose(TempStudent);

End;

StudentList := Nil;

End;

Procedure DeleteStudent(Var StudentList: PStudent;

Const StudentData: TStudentData);

Var

CurrStudent, TempStudent: PStudent;

Begin

CurrStudent := StudentList;

If IsStudentDataEqual(CurrStudent^.Data, StudentData) Then

Begin

StudentList := CurrStudent^.Next;

Dispose(CurrStudent);

End

Else

Begin

While Not IsStudentDataEqual(CurrStudent^.Next^.Data, StudentData) Do

CurrStudent := CurrStudent^.Next;

TempStudent := CurrStudent^.Next;

CurrStudent^.Next := CurrStudent^.Next^.Next;

Dispose(TempStudent);

End;

End;

Procedure DeleteStudentsInGroup(GroupNumber: Integer;

Var StudentList: PStudent);

Var

CurrStudent, TempStudent: PStudent;

Begin

CurrStudent := StudentList;

If CurrStudent <> Nil Then

Begin

If (CurrStudent^.Data.GroupNumber = GroupNumber) Then

Begin

StudentList := CurrStudent^.Next;

Dispose(CurrStudent);

End

Else

Begin

While CurrStudent <> Nil Do

Begin

If CurrStudent^.Next^.Data.GroupNumber = GroupNumber Then

Begin

CurrStudent := CurrStudent^.Next;

TempStudent := CurrStudent^.Next;

CurrStudent^.Next := CurrStudent^.Next^.Next;

Dispose(TempStudent);

End;

End;

End;

End;

End;

Function CreateStudent(Const StudentData: TStudentData): PStudent;

Var

NewStudent: PStudent;

Begin

New(NewStudent);

NewStudent^.Data := StudentData;

NewStudent^.Next := Nil;

CreateStudent := NewStudent;

End;

Procedure AddStudent(Var StudentList: PStudent;

Const StudentData: TStudentData);

Var

NewStudent, CurrStudent: PStudent;

Begin

NewStudent := CreateStudent(StudentData);

If StudentList = Nil Then

StudentList := NewStudent

Else

Begin

CurrStudent := StudentList;

While CurrStudent^.Next <> Nil Do

CurrStudent := CurrStudent^.Next;

CurrStudent^.Next := NewStudent;

End;

End;

Function IsStudentDataEqual(Const StudentData1, StudentData2

: TStudentData): Boolean;

Var

IsEqual: Boolean;

Begin

IsEqual := (StudentData1.GroupNumber = StudentData2.GroupNumber) And

(StudentData1.Name = StudentData2.Name) And

(StudentData1.Surname = StudentData2.Surname) And

(StudentData1.Patronymic = StudentData2.Patronymic) And

(StudentData1.Marks[Logic] = StudentData2.Marks[Logic]) And

(StudentData1.Marks[Algebra] = StudentData2.Marks[Algebra]) And

(StudentData1.Marks[DM] = StudentData2.Marks[DM]) And

(StudentData1.Marks[History] = StudentData2.Marks[History]) And

(StudentData1.Marks[BAAP] = StudentData2.Marks[BAAP]) And

(StudentData1.Marks[BPE] = StudentData2.Marks[BPE]);

IsStudentDataEqual := IsEqual;

End;

Function IsExistStudent(Const StudentList: PStudent;

Const StudentData: TStudentData): Boolean;

Var

CurrStudent: PStudent;

IsExist: Boolean;

Begin

CurrStudent := StudentList;

IsExist := False;

If CurrStudent <> Nil Then

Repeat

IsExist := IsStudentDataEqual(CurrStudent^.Data, StudentData);

CurrStudent := CurrStudent^.Next

Until (IsExist) Or (CurrStudent = Nil);

IsExistStudent := IsExist;

End;

Procedure DrawMarksGrid(Var MarkGrid: TStringGrid);

Begin

With MarkGrid Do

Begin

Height := (DefaultRowHeight \* 2) + 6;

Cells[0, 0] := 'Логика';

Cells[1, 0] := 'ЛАиАГ';

Cells[2, 0] := 'История';

ColWidths[2] := 130;

Cells[3, 0] := 'ДМ';

ColWidths[3] := 70;

Cells[4, 0] := 'ОАиП';

Cells[5, 0] := 'ОПИ';

ScrollBars := TScrollStyle.SsNone;

End;

End;

Function IsStudentDataEqualNSP(Const StudentData1: TStudentData;

Name, Surname, Patronymic: TNameString): Boolean;

Var

IsEqual: Boolean;

Begin

IsEqual := (StudentData1.Name = Name) And (StudentData1.Surname = Surname)

And (StudentData1.Patronymic = Patronymic);

IsStudentDataEqualNSP := IsEqual;

End;

Function IsExistStudentNSP(Const StudentList: PStudent;

Name, Surname, Patronymic: TNameString): Boolean;

Var

CurrStudent: PStudent;

IsExist: Boolean;

Begin

CurrStudent := StudentList;

IsExist := False;

If CurrStudent <> Nil Then

Repeat

IsExist := IsStudentDataEqualNSP(CurrStudent^.Data, Name, Surname,

Patronymic);

CurrStudent := CurrStudent^.Next

Until (IsExist) Or (CurrStudent = Nil);

IsExistStudentNSP := IsExist;

End;

Function SearchStudentNSP(StudentList: PStudent;

Name, Surname, Patronymic: TNameString): PStudent;

Var

NewStudent: PStudent;

Begin

NewStudent := StudentList;

If IsExistStudentNSP(StudentList, Name, Surname, Patronymic) Then

Begin

While Not IsStudentDataEqualNSP(NewStudent^.Data, Name, Surname,

Patronymic) Do

NewStudent := NewStudent^.Next;

SearchStudentNSP := NewStudent;

End;

End;

Procedure ClearStudentGrid(Var StudentsGrid: TStringGrid);

Begin

With StudentsGrid Do

Begin

RowCount := 1;

ColCount := 11;

Cells[0, 0] := 'Номер Группы';

Cells[1, 0] := 'Фамилия';

Cells[2, 0] := 'Имя';

Cells[3, 0] := 'Отчество';

Cells[4, 0] := 'Логика';

Cells[5, 0] := 'ЛАиАГ';

Cells[6, 0] := 'История';

Cells[7, 0] := 'ДМ';

Cells[8, 0] := 'ОАиП';

Cells[9, 0] := 'ОПИ';

Cells[10, 0] := 'Средний балл';

ColWidths[0] := 150;

ColWidths[1] := 160;

ColWidths[2] := 180;

ColWidths[3] := 170;

ColWidths[10] := 170;

RowCount := 1;

End;

End;

Function FindDeptStudents(StudentsList: PStudent; CurrDicipline: Integer)

: PStudent;

Var

CurrStudent: PStudent;

CurrDic: Diciplines;

NewStudent: PStudent;

Begin

CurrStudent := StudentsList;

NewStudent := Nil;

Case CurrDicipline Of

1:

CurrDic := Logic;

2:

CurrDic := Algebra;

3:

CurrDic := History;

4:

CurrDic := DM;

5:

CurrDic := BAAP;

6:

CurrDic := BPE;

End;

While CurrStudent <> Nil Do

Begin

If CurrStudent^.Data.Marks[CurrDic] < 4 Then

AddStudent(NewStudent, CurrStudent^.Data);

CurrStudent := CurrStudent^.Next;

End;

FindDeptStudents := NewStudent;

End;

Procedure DrawStudentsInGroupGrid(Var StudentsInGroupGrid: TStringGrid;

StudentsList: PStudent; CurrentGroup: Integer);

Var

CurrStudent: PStudent;

CurrRow: Integer;

Begin

With StudentsInGroupGrid Do

Begin

ColCount := 11;

Cells[0, 0] := 'Номер Группы';

Cells[1, 0] := 'Фамилия';

Cells[2, 0] := 'Имя';

Cells[3, 0] := 'Отчество';

Cells[4, 0] := 'Логика';

Cells[5, 0] := 'ЛАиАГ';

Cells[6, 0] := 'История';

Cells[7, 0] := 'ДМ';

Cells[8, 0] := 'ОАиП';

Cells[9, 0] := 'ОПИ';

Cells[10, 0] := 'Средний балл';

ColWidths[0] := 150;

ColWidths[1] := 160;

ColWidths[2] := 180;

ColWidths[3] := 170;

ColWidths[10] := 170;

End;

CurrStudent := StudentsList;

CurrRow := 1;

If (CurrentGroup = 0) Or (CurrentGroup = 1) Then

Begin

If CurrentGroup = 1 Then

Begin

While CurrStudent <> Nil Do

Begin

With StudentsInGroupGrid, CurrStudent^.Data Do

Begin

Cells[0, CurrRow] := IntToStr(GroupNumber);

Cells[1, CurrRow] := String(Surname);

Cells[2, CurrRow] := String(Name);

Cells[3, CurrRow] := String(Patronymic);

Cells[4, CurrRow] := IntToStr(Marks[Logic]);

Cells[5, CurrRow] := IntToStr(Marks[Algebra]);

Cells[6, CurrRow] := IntToStr(Marks[History]);

Cells[7, CurrRow] := IntToStr(Marks[DM]);

Cells[8, CurrRow] := IntToStr(Marks[BAAP]);

Cells[9, CurrRow] := IntToStr(Marks[BPE]);

Cells[10, CurrRow] := FloatToStr(AvgMark);

Inc(CurrRow);

End;

CurrStudent := CurrStudent^.Next;

End;

StudentsInGroupGrid.RowCount := CurrRow;

If StudentsInGroupGrid.RowCount > 10 Then

Begin

StudentsInGroupGrid.ScrollBars := TScrollStyle.SsVertical;

StudentsInGroupGrid.Height := 400

End

Else

Begin

StudentsInGroupGrid.ScrollBars := TScrollStyle.SsNone;

StudentsInGroupGrid.Height :=

(StudentsInGroupGrid.DefaultRowHeight +

StudentsInGroupGrid.GridLineWidth) \*

StudentsInGroupGrid.RowCount + 5;

End;

End

Else

Begin

If CurrStudent <> Nil Then

Begin

With StudentsInGroupGrid, CurrStudent^.Data Do

Begin

Cells[0, CurrRow] := IntToStr(GroupNumber);

Cells[1, CurrRow] := String(Surname);

Cells[2, CurrRow] := String(Name);

Cells[3, CurrRow] := String(Patronymic);

Cells[4, CurrRow] := IntToStr(Marks[Logic]);

Cells[5, CurrRow] := IntToStr(Marks[Algebra]);

Cells[6, CurrRow] := IntToStr(Marks[History]);

Cells[7, CurrRow] := IntToStr(Marks[DM]);

Cells[8, CurrRow] := IntToStr(Marks[BAAP]);

Cells[9, CurrRow] := IntToStr(Marks[BPE]);

Cells[10, CurrRow] := FloatToStr(AvgMark);

RowCount := 2;

End;

End;

End;

End

Else

Begin

While CurrStudent <> Nil Do

Begin

With StudentsInGroupGrid, CurrStudent^.Data Do

Begin

If GroupNumber = CurrentGroup Then

Begin

Cells[0, CurrRow] := IntToStr(GroupNumber);

Cells[1, CurrRow] := String(Surname);

Cells[2, CurrRow] := String(Name);

Cells[3, CurrRow] := String(Patronymic);

Cells[4, CurrRow] := IntToStr(Marks[Logic]);

Cells[5, CurrRow] := IntToStr(Marks[Algebra]);

Cells[6, CurrRow] := IntToStr(Marks[History]);

Cells[7, CurrRow] := IntToStr(Marks[DM]);

Cells[8, CurrRow] := IntToStr(Marks[BAAP]);

Cells[9, CurrRow] := IntToStr(Marks[BPE]);

Cells[10, CurrRow] := FloatToStr(AvgMark);

Inc(CurrRow);

End;

End;

CurrStudent := CurrStudent^.Next;

End;

If StudentsInGroupGrid.RowCount > 10 Then

Begin

StudentsInGroupGrid.ScrollBars := TScrollStyle.SsVertical;

StudentsInGroupGrid.Height := 400

End

Else

Begin

StudentsInGroupGrid.ScrollBars := TScrollStyle.SsNone;

StudentsInGroupGrid.Height :=

(StudentsInGroupGrid.DefaultRowHeight +

StudentsInGroupGrid.GridLineWidth) \*

StudentsInGroupGrid.RowCount + 5;

End;

End;

End;

End.

Unit ViewGroupListUnit;

Interface

Uses

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

System.Classes, Vcl.Graphics,

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

GroupListActionUnit, GroupsLinkedListUnit, Vcl.Menus, Vcl.ExtDlgs;

Type

TGroupListForm = Class(TForm)

AddGroup: TButton;

DeleteGroup: TButton;

EditGroup: TButton;

GroupGrid: TStringGrid;

ViewStudentsInGroupButton: TButton;

GroupMainMenu: TMainMenu;

FileDialog: TMenuItem;

OpenFile: TMenuItem;

SaveFile: TMenuItem;

OpenTextFileDialog: TOpenTextFileDialog;

SaveTextFileDialog: TSaveTextFileDialog;

Procedure CreateParams(Var Params: TCreateParams); Override;

Procedure AddGroupClick(Sender: TObject);

Procedure DeleteGroupClick(Sender: TObject);

Procedure EditGroupClick(Sender: TObject);

Procedure FormShow(Sender: TObject);

Procedure ViewStudentsInGroupButtonClick(Sender: TObject);

Procedure GroupGridSelectCell(Sender: TObject; ACol, ARow: Integer;

Var CanSelect: Boolean);

Procedure OpenFileClick(Sender: TObject);

Procedure SaveFileClick(Sender: TObject);

Private

{ Private declarations }

Public

{ Public declarations }

End;

Var

GroupListForm: TGroupListForm;

OldGroupData: TGroupData;

Implementation

Uses

MainMenuUnit, ViewStudentsInGroupUnit, FileUnit, StudentLinkedListUnit;

{$R \*.dfm}

Procedure TGroupListForm.CreateParams(Var Params: TCreateParams);

Begin

Inherited;

Params.ExStyle := Params.ExStyle Or WS\_EX\_APPWINDOW;

End;

Procedure CreateForm(FormClass: TFormClass);

Var

Form: TForm;

Begin

GroupListForm.Visible := False;

Form := FormClass.Create(MainMenuForm);

Form.Icon := MainMenuForm.Icon;

Form.ShowModal;

GroupListForm.Visible := True;

End;

Procedure CreateActionForm(Const TitleLabelCaption, ActionButtonCaption

: String);

Begin

GroupListActionForm := TGroupListActionForm.Create(GroupListActionForm);

GroupListActionForm.Icon := GroupListForm.Icon;

GroupListActionForm.TitleLabel.Width := 200;

GroupListActionForm.TitleLabel.Caption := TitleLabelCaption;

GroupListActionForm.ActionButton.Caption := ActionButtonCaption;

GroupListActionForm.ShowModal;

End;

Procedure TGroupListForm.AddGroupClick(Sender: TObject);

Begin

StateOfGForm := AddG;

CreateActionForm('Добавление группы', 'Добавить');

If IsEdited Then

Begin

GroupGrid.RowCount := GroupGrid.RowCount + 1;

DrawGroupGrid(GroupGrid, GroupList);

IsEdited := False;

IsSaved := False;

SaveFile.Enabled := True;

End;

End;

Procedure TGroupListForm.ViewStudentsInGroupButtonClick(Sender: TObject);

Begin

CreateForm(TViewStudentsInGroupForm);

End;

Function GetGroupDataFromGrid(GroupGrid: TStringGrid): TGroupData;

Var

OldGroupData: TGroupData;

Begin

With GroupGrid, OldGroupData Do

Begin

GroupNumber := StrToInt(Cells[0, Row]);

Year := StrToInt(Cells[1, Row]);

Code := Cells[2, Row];

CountOfStudents := StrToInt(Cells[3, Row]);

End;

GetGroupDataFromGrid := OldGroupData;

End;

Procedure TGroupListForm.DeleteGroupClick(Sender: TObject);

Var

Confirmation, GroupCount: Integer;

Begin

If GroupGrid.Row > 0 Then

Begin

Confirmation := Application.MessageBox

('Вы действительно хотите удалить группу?', 'Удаление кандидата',

MB\_YESNO + MB\_ICONQUESTION + MB\_DEFBUTTON2);

If Confirmation = IDYES Then

Begin

DeleteGroupF(GroupList, GetGroupDataFromGrid(GroupGrid),

StudentsList);

GroupGrid.RowCount := GroupGrid.RowCount - 1;

DrawGroupGrid(GroupGrid, GroupList);

GroupCount := CountGroups(GroupList);

IsSaved := GroupCount = 0;

SaveFile.Enabled := GroupCount > 0;

End;

End

Else

Application.MessageBox('Не выбрано редактируемое поле!',

'Предупреждение', MB\_OK + MB\_ICONWARNING);

End;

Procedure TGroupListForm.EditGroupClick(Sender: TObject);

Begin

If GroupGrid.Row > 0 Then

Begin

StateOfGForm := EditG;

OldGroupData := GetGroupDataFromGrid(GroupGrid);

CreateActionForm('Изменение группы', 'Изменить');

If IsEdited Then

Begin

DrawGroupGrid(GroupGrid, GroupList);

IsEdited := False;

IsSaved := False;

SaveFile.Enabled := True;

End;

End

Else

Application.MessageBox('Не выбрано редактируемое поле!',

'Предупреждение', MB\_OK + MB\_ICONWARNING);

End;

Procedure TGroupListForm.FormShow(Sender: TObject);

Var

GroupCount: Integer;

Begin

GroupCount := CountGroups(GroupList);

GroupGrid.RowCount := CountGroups(GroupList) + 1;

SaveFile.Enabled := GroupCount > 0;

DrawGroupGrid(GroupGrid, GroupList);

End;

Procedure TGroupListForm.GroupGridSelectCell(Sender: TObject;

ACol, ARow: Integer; Var CanSelect: Boolean);

Begin

ViewStudentsInGroupButton.Enabled := Not(ARow = 0);

If ARow > 0 Then

SelectedGroup := StrToInt(GroupGrid.Cells[0, Arow]);

End;

Procedure TGroupListForm.OpenFileClick(Sender: TObject);

Var

InputGroupFile: TGroupDataFile;

InputStudentFile: TStudentDataFile;

TempGroupList: PGroup;

TempStudent: PStudent;

IsCorrect: Boolean;

Begin

OpenTextFileDialog.DefaultExt := 'grplst';

If OpenTextFileDialog.Execute Then

Begin

AssignFile(InputGroupFile, OpenTextFileDialog.FileName);

TempGroupList := Nil;

IsCorrect := ReadFileData(InputGroupFile, InputStudentFile,

TempGroupList, TempStudent, True);

If IsCorrect Then

Begin

ClearGroups(GroupList);

End

Else

Begin

ClearGroups(TempGroupList);

Application.MessageBox('Содержимое файла повреждено!', 'Ошибка',

MB\_OK + MB\_ICONERROR);

End;

End;

If IsCorrect Then

Begin

OpenTextFileDialog.DefaultExt := 'stdlst';

If OpenTextFileDialog.Execute Then

Begin

TempStudent := Nil;

AssignFile(InputStudentFile, OpenTextFileDialog.FileName);

IsCorrect := ReadFileData(InputGroupFile, InputStudentFile,

TempGroupList, TempStudent, False);

If IsCorrect Then

Begin

ClearStudents(StudentsList);

RecordFileData(GroupList, TempGroupList, StudentsList,

TempStudent, GroupGrid);

IsSaved := True;

SaveFile.Enabled := True;

End

Else

Begin

ClearStudents(TempStudent);

Application.MessageBox('Содержимое файла повреждено!', 'Ошибка',

MB\_OK + MB\_ICONERROR);

End;

End;

End;

End;

Procedure TGroupListForm.SaveFileClick(Sender: TObject);

Var

OutputGroupFile: TGroupDataFile;

OutputStudentFile: TStudentDataFile;

Begin

SaveTextFileDialog.DefaultExt := 'grplst';

If SaveTextFileDialog.Execute Then

Begin

AssignFile(OutputGroupFile, SaveTextFileDialog.FileName);

WriteFileData(OutputGroupFile, OutputStudentFile, GroupList,

StudentsList, True);

End;

SaveTextFileDialog.DefaultExt := 'stdlst';

If SaveTextFileDialog.Execute Then

Begin

AssignFile(OutputStudentFile, SaveTextFileDialog.FileName);

WriteFileData(OutputGroupFile, OutputStudentFile, GroupList,

StudentsList, False);

IsSaved := True;

End;

End;

End.

Unit ViewStudentsInGroupUnit;

Interface

Uses

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

System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls, Vcl.Grids, Math,

StudentsInGroupActionUnit, StudentLinkedListUnit;

Type

TViewStudentsInGroupForm = Class(TForm)

StudentInGroupGrid: TStringGrid;

AddStudentButton: TButton;

DeleteStudentButton: TButton;

EditStudentButton: TButton;

Procedure CreateParams(Var Params: TCreateParams); Override;

Procedure AddStudentButtonClick(Sender: TObject);

Procedure FormShow(Sender: TObject);

Procedure DeleteStudentButtonClick(Sender: TObject);

Procedure EditStudentButtonClick(Sender: TObject);

Private

{ Private declarations }

Public

{ Public declarations }

End;

Var

ViewStudentsInGroupForm: TViewStudentsInGroupForm;

OldStudentData: TStudentData;

Implementation

Uses

MainMenuUnit, GroupsLinkedListUnit, ViewGroupListUnit;

{$R \*.dfm}

Procedure TViewStudentsInGroupForm.CreateParams(Var Params: TCreateParams);

Begin

Inherited;

Params.ExStyle := Params.ExStyle Or WS\_EX\_APPWINDOW;

End;

Function GetStudentDataFromGrid(StudentsInGroupGrid: TStringGrid): TStudentData;

Var

OldStudentData: TStudentData;

Begin

With StudentsInGroupGrid, OldStudentData Do

Begin

GroupNumber := StrToInt(Cells[0, Row]);

OldStudentData.Name := Cells[2, Row];

Surname := Cells[1, Row];

Patronymic := Cells[3, Row];

Marks[Logic] := StrToInt(Cells[4, Row]);

Marks[Algebra] := StrToInt(Cells[5, Row]);

Marks[History] := StrToInt(Cells[6, Row]);

Marks[DM] := StrToInt(Cells[7, Row]);

Marks[BAAP] := StrToInt(Cells[8, Row]);

Marks[BPE] := StrToInt(Cells[9, Row]);

AvgMark :=

RoundTo(((Marks[Logic] + Marks[Algebra] + Marks[History] + Marks[DM] +

Marks[BAAP] + Marks[BPE]) / 6), -2);

End;

GetStudentDataFromGrid := OldStudentData;

End;

Procedure TViewStudentsInGroupForm.DeleteStudentButtonClick(Sender: TObject);

Var

Confirmation, StudentsCount: Integer;

Begin

StateOfSForm := DeleteS;

If StudentInGroupGrid.Row > 0 Then

Begin

Confirmation := Application.MessageBox

('Вы действительно хотите удалить студента?', 'Удаление кандидата',

MB\_YESNO + MB\_ICONQUESTION + MB\_DEFBUTTON2);

If Confirmation = IDYES Then

Begin

DeleteStudent(StudentsList,

GetStudentDataFromGrid(StudentInGroupGrid));

StudentInGroupGrid.RowCount := StudentInGroupGrid.RowCount - 1;

DrawStudentsInGroupGrid(StudentInGroupGrid, StudentsList,

SelectedGroup);

StudentsCount := FindCountInGroup(GroupList, SelectedGroup);

DeleteOneStudent(GroupList, SelectedGroup);

GroupListForm.SaveFile.Enabled := (StudentsCount > 0);

End;

End

Else

Application.MessageBox('Не выбрано редактируемое поле!',

'Предупреждение', MB\_OK + MB\_ICONWARNING);

End;

Procedure CreateActionForm(Const TitleLabelCaption, ActionButtonCaption

: String);

Begin

StudentsInGroupActionForm := TStudentsInGroupActionForm.Create

(StudentsInGroupActionForm);

StudentsInGroupActionForm.Icon := ViewStudentsInGroupForm.Icon;

StudentsInGroupActionForm.TitleLabel.Width := 200;

StudentsInGroupActionForm.TitleLabel.Caption := TitleLabelCaption;

StudentsInGroupActionForm.ActionButton.Caption := ActionButtonCaption;

StudentsInGroupActionForm.ShowModal;

End;

Procedure TViewStudentsInGroupForm.EditStudentButtonClick(Sender: TObject);

Begin

If StudentInGroupGrid.Row > 0 Then

Begin

StateOfSForm := EditS;

OldStudentData := GetStudentDataFromGrid(StudentInGroupGrid);

CreateActionForm('Изменение информации', 'Изменить');

If IsEdited Then

Begin

DrawStudentsInGroupGrid(StudentInGroupGrid, StudentsList,

SelectedGroup);

IsEdited := False;

IsSaved := False;

GroupListForm.SaveFile.Enabled := True;

// ContentEdit.Text := '';

End;

End

Else

Application.MessageBox('Не выбрано редактируемое поле!',

'Предупреждение', MB\_OK + MB\_ICONWARNING);

End;

Procedure TViewStudentsInGroupForm.FormShow(Sender: TObject);

Var

StudentsCount: Integer;

Begin

StudentsCount := FindCountInGroup(GroupList, SelectedGroup);

StudentInGroupGrid.RowCount := StudentsCount + 1;

GroupListForm.SaveFile.Enabled := (StudentsCount > 0);

DrawStudentsInGroupGrid(StudentInGroupGrid, StudentsList, SelectedGroup);

End;

Procedure TViewStudentsInGroupForm.AddStudentButtonClick(Sender: TObject);

Begin

StateOfSForm := AddS;

CreateActionForm('Добавление студента', 'Добавить');

If IsEdited Then

Begin

StudentInGroupGrid.RowCount := FindCountInGroup(GroupList,

SelectedGroup) + 1;

DrawStudentsInGroupGrid(StudentInGroupGrid, StudentsList,

SelectedGroup);

End;

End;

End.

Unit StudentsInGroupActionUnit;

Interface

Uses

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

System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls, Vcl.Grids, MainMenuUnit,

Math;

Type

TStudentsInGroupActionForm = Class(TForm)

TitleLabel: TLabel;

NameLabel: TLabel;

SurnameLabel: TLabel;

PatronymicLabel: TLabel;

NameEdit: TEdit;

SurnameEdit: TEdit;

PatronymicEdit: TEdit;

MarkGrid: TStringGrid;

ActionButton: TButton;

CancelButton: TButton;

MarksLabel: TLabel;

GroupNumberLabel: TLabel;

GroupNumberEdit: TEdit;

Procedure CreateParams(Var Params: TCreateParams); Override;

Procedure CancelButtonClick(Sender: TObject);

Procedure FormShow(Sender: TObject);

Procedure MarkGridKeyPress(Sender: TObject; Var Key: Char);

Procedure ActionButtonClick(Sender: TObject);

Procedure EditChange(Sender: TObject);

Procedure EditContextPopup(Sender: TObject; MousePos: TPoint;

Var Handled: Boolean);

Procedure EditKeyPress(Sender: TObject; Var Key: Char);

Procedure EditKeyDown(Sender: TObject; Var Key: Word;

Shift: TShiftState);

Procedure MarkGridSetEditText(Sender: TObject; ACol, ARow: Integer;

Const Value: String);

Procedure GroupNumberEditKeyPress(Sender: TObject; Var Key: Char);

Private

{ Private declarations }

Public

{ Public declarations }

End;

Var

StudentsInGroupActionForm: TStudentsInGroupActionForm;

Implementation

{$R \*.dfm}

Uses

StudentLinkedListUnit, InputEditUnit, GroupsLinkedListUnit,

ViewStudentsInGroupUnit;

Procedure TStudentsInGroupActionForm.CreateParams(Var Params: TCreateParams);

Begin

Inherited;

Params.ExStyle := Params.ExStyle Or WS\_EX\_APPWINDOW;

End;

Procedure SetStudentData(OldStudentData: TStudentData);

Begin

With StudentsInGroupActionForm, OldStudentData, MarkGrid Do

Begin

NameEdit.Text := String(OldStudentData.Name);

SurnameEdit.Text := String(Surname);

PatronymicEdit.Text := String(Patronymic);

// GroupNumberEdit.Text := IntToStr(GroupNumber);

GroupNumberEdit.Text := IntToStr(SelectedGroup);

Cells[0, 1] := IntToStr(Marks[Logic]);

Cells[1, 1] := IntToStr(Marks[Algebra]);

Cells[2, 1] := IntToStr(Marks[History]);

Cells[3, 1] := IntToStr(Marks[DM]);

Cells[4, 1] := IntToStr(Marks[BAAP]);

Cells[5, 1] := IntToStr(Marks[BPE]);

End;

End;

Function CreateStudentData(): TStudentData;

Var

NewStudent: TStudentData;

I: Integer;

Begin

With NewStudent, StudentsInGroupActionForm Do

Begin

GroupNumber := StrToInt(GroupNumberEdit.Text);

NewStudent.Name := String(NameEdit.Text);

Surname := String(SurnameEdit.Text);

Patronymic := String(PatronymicEdit.Text);

With MarkGrid Do

Begin

Marks[Logic] := StrToInt(Cells[0, 1]);

Marks[Algebra] := StrToInt(Cells[1, 1]);

Marks[History] := StrToInt(Cells[2, 1]);

Marks[DM] := StrToInt(Cells[3, 1]);

Marks[BAAP] := StrToInt(Cells[4, 1]);

Marks[BPE] := StrToInt(Cells[5, 1]);

End;

AvgMark :=

RoundTo(((Marks[Logic] + Marks[Algebra] + Marks[History] + Marks[DM] +

Marks[BAAP] + Marks[BPE]) / 6), -2);

End;

CreateStudentData := NewStudent;

End;

Procedure TStudentsInGroupActionForm.ActionButtonClick(Sender: TObject);

Var

NewStudentData: TStudentData;

Begin

NewStudentData := CreateStudentData();

If StateOfSForm = AddS Then

Begin

AddStudent(StudentsList, NewStudentData);

AddStudentToGroup(GroupList, SelectedGroup);

End

Else If StateOfSForm = EditS Then

Begin

EditStudent(GroupList, OldStudentData, NewStudentData, StudentsList);

End;

IsEdited := True;

Close;

End;

Procedure TStudentsInGroupActionForm.CancelButtonClick(Sender: TObject);

Begin

IsEdited := False;

Close;

End;

Procedure TStudentsInGroupActionForm.FormShow(Sender: TObject);

Begin

If StateOfSForm = AddS Then

Begin

GroupNumberEdit.Text := IntToStr(SelectedGroup);

GroupNumberEdit.Visible := False;

GroupNumberLabel.Visible := False;

End;

If StateOfSForm = EditS Then

Begin

GroupNumberEdit.Text := IntToStr(SelectedGroup);

GroupNumberEdit.Visible := True;

GroupNumberLabel.Visible := True;

SetStudentData(OldStudentData);

End;

DrawMarksGrid(MarkGrid);

End;

Procedure TStudentsInGroupActionForm.GroupNumberEditKeyPress(Sender: TObject;

Var Key: Char);

Begin

CheckLength(GROUP\_LENGTH, GroupNumberEdit, Key);

CheckComboButtons(Key, GroupNumberEdit, NUMBERS);

End;

Procedure TStudentsInGroupActionForm.MarkGridKeyPress(Sender: TObject;

Var Key: Char);

Begin

CheckValidMarks(MarkGrid, Key);

End;

Procedure TStudentsInGroupActionForm.MarkGridSetEditText(Sender: TObject;

ACol, ARow: Integer; Const Value: String);

Begin

ActionButton.Enabled := IsFullFieldsStudents(NameEdit, SurnameEdit,

PatronymicEdit, GroupNumberEdit, MarkGrid) And

(FindCountInGroup(GroupList, SelectedGroup) < 40) And

IsExistGroupByNumber(GroupList, StrToInt(GroupNumberEdit.Text));

End;

Procedure TStudentsInGroupActionForm.EditKeyDown(Sender: TObject; Var Key: Word;

Shift: TShiftState);

Begin

CheckShiftAndArrows(Key, Shift);

End;

Procedure TStudentsInGroupActionForm.EditKeyPress(Sender: TObject;

Var Key: Char);

Var

Edit: TEdit;

Begin

Edit := TEdit(Sender);

NameComponentKeyPress(Edit.Text, Edit.SelStart, Edit.SelLength,

NAME\_LENGTH, Key);

End;

Procedure TStudentsInGroupActionForm.EditContextPopup(Sender: TObject;

MousePos: TPoint; Var Handled: Boolean);

Begin

Handled := True;

End;

Procedure TStudentsInGroupActionForm.EditChange(Sender: TObject);

Var

NewStudentData: TStudentData;

CanPress: Boolean;

Begin

ActionButton.Enabled := False;

If IsFullFieldsStudents(NameEdit, SurnameEdit, PatronymicEdit,

GroupNumberEdit, MarkGrid) Then

Begin

NewStudentData := CreateStudentData();

CanPress := (Not IsExistStudent(StudentsList, NewStudentData)) And

IsExistGroupByNumber(GroupList, StrToInt(GroupNumberEdit.Text));

ActionButton.Enabled := CanPress;

End;

End;

End.

Unit FindStudentUnit;

Interface

Uses

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

System.Classes, Vcl.Graphics,

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

Type

TFindStudentForm = Class(TForm)

TitleLabel: TLabel;

NameLabel: TLabel;

SurnameLabel: TLabel;

PatronymicLabel: TLabel;

NameEdit: TEdit;

SurnameEdit: TEdit;

PatronymicEdit: TEdit;

FindStudentGrid: TStringGrid;

SearchButton: TButton;

Procedure FormCreate(Sender: TObject);

Procedure CreateParams(Var Params: TCreateParams); Override;

Procedure SearchButtonClick(Sender: TObject);

Procedure EditKeyDown(Sender: TObject; Var Key: Word;

Shift: TShiftState);

Procedure EditKeyPress(Sender: TObject; Var Key: Char);

Procedure EditOnChange(Sender: TObject);

Private

{ Private declarations }

Public

{ Public declarations }

End;

Var

FindStudentForm: TFindStudentForm;

Implementation

{$R \*.dfm}

USes

MainMenuUnit, StudentLinkedListUnit, InputEditUnit;

Procedure TFindStudentForm.CreateParams(Var Params: TCreateParams);

Begin

Inherited;

Params.ExStyle := Params.ExStyle Or WS\_EX\_APPWINDOW;

End;

Procedure TFindStudentForm.EditKeyDown(Sender: TObject; Var Key: Word;

Shift: TShiftState);

Begin

CheckShiftAndArrows(Key, Shift);

End;

Procedure TFindStudentForm.EditKeyPress(Sender: TObject; Var Key: Char);

Var

Edit: TEdit;

Begin

Edit := TEdit(Sender);

NameComponentKeyPress(Edit.Text, Edit.SelStart, Edit.SelLength,

NAME\_LENGTH, Key);

End;

Procedure TFindStudentForm.EditOnChange(Sender: TObject);

Var

NewStudentData: TStudentData;

CanPress: Boolean;

Begin

SearchButton.Enabled := False;

If IsFullFieldsStudentsNSP(NameEdit, SurnameEdit, PatronymicEdit) Then

Begin

CanPress := (IsExistStudentNSP(StudentsList, Trim(NameEdit.Text),

Trim(SurnameEdit.Text), Trim(PatronymicEdit.Text)));

SearchButton.Enabled := CanPress;

End;

End;

Procedure TFindStudentForm.FormCreate(Sender: TObject);

Begin

FindStudentGrid.RowCount := 1;

DrawStudentsInGroupGrid(FindStudentGrid, Nil, 0);

End;

Procedure TFindStudentForm.SearchButtonClick(Sender: TObject);

Var

FoundStudent: PStudent;

Begin

FoundStudent := SearchStudentNSP(StudentsList, Trim(NameEdit.Text),

Trim(SurnameEdit.Text), Trim(PatronymicEdit.Text));

FindStudentGrid.RowCount := FindStudentGrid.RowCount + 1;

DrawStudentsInGroupGrid(FindStudentGrid, FoundStudent, 0)

End;

End.

Unit ListsActionUnit;

Interface

Uses

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

System.Classes, Vcl.Graphics,

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

Type

TListsActionForm = Class(TForm)

FindStudentGrid: TStringGrid;

TitleLabel: TLabel;

DiciplineComboBox: TComboBox;

ChooseLabel: TLabel;

Procedure CreateParams(Var Params: TCreateParams); Override;

Procedure FormCreate(Sender: TObject);

Procedure DiciplineComboBoxChange(Sender: TObject);

Private

{ Private declarations }

Public

{ Public declarations }

End;

Var

ListsActionForm: TListsActionForm;

Implementation

{$R \*.dfm}

Uses

MainMenuUnit, StudentLinkedListUnit;

Procedure TListsActionForm.CreateParams(Var Params: TCreateParams);

Begin

Inherited;

Params.ExStyle := Params.ExStyle Or WS\_EX\_APPWINDOW;

End;

Procedure TListsActionForm.DiciplineComboBoxChange(Sender: TObject);

Var

CurrentDicipline: Integer;

NewStudent: PStudent;

Begin

ClearStudentGrid(FindStudentGrid);

CurrentDicipline := DiciplineComboBox.ItemIndex;

NewStudent := FindDeptStudents(StudentsList, CurrentDicipline + 1);

If NewStudent <> Nil Then

DrawStudentsInGroupGrid(FindStudentGrid, NewStudent, 1)

Else

Begin

Application.MessageBox('Студентов должников нет :(', 'Информация',

MB\_OK + MB\_ICONINFORMATION);

ClearStudentGrid(FindStudentGrid);

End;

End;

Procedure TListsActionForm.FormCreate(Sender: TObject);

Var

NewStudentList: PStudent;

Begin

DiciplineComboBox.Items.Add('Логика');

DiciplineComboBox.Items.Add('ЛАиАГ');

DiciplineComboBox.Items.Add('История');

DiciplineComboBox.Items.Add('ДМ');

DiciplineComboBox.Items.Add('ОАиП');

DiciplineComboBox.Items.Add('ОПИ');

If StateListForm = Expel Then

Begin

ChooseLabel.Visible := False;

DiciplineComboBox.Visible := False;

NewStudentList := FindExpellStudents(StudentsList);

If NewStudentList <> Nil Then

Begin

DrawStudentsInGroupGrid(FindStudentGrid, NewStudentList, 1);

End

Else

Begin

Application.MessageBox('Студентов на отчилсение нет :(',

'Информация', MB\_OK + MB\_ICONINFORMATION);

Close;

End;

End;

If StateListForm = Debt Then

Begin

DrawStudentsInGroupGrid(FindStudentGrid, Nil, 1);

End;

End;

End.

Unit FileUnit;

Interface

Uses

GroupsLinkedListUnit, StudentLinkedListUnit, SysUtils, Vcl.Grids;

Type

TGroupDataFile = File Of TGroupData;

TStudentDataFile = File Of TStudentData;

Function IsCorrectFileData(Const TempGroupData: TGroupData;

Const TempStudentData: TStudentData; Const IsGroup: Boolean): Boolean;

Function ReadFileData(Var InputGroupFile: TGroupDataFile;

Var InputStudentsFile: TStudentDataFile; Var TempGroupList: PGroup; Var TempStudentList: PStudent; IsGroup: Boolean): Boolean;

Procedure WriteFileData(Var OutputGroupFile: TGroupDataFile;

Var OutputStudentFile: TStudentDataFile; Const GroupList: PGroup;

Const StudentList: PStudent; IsGroup: Boolean);

Procedure RecordFileData(Var GroupList: PGroup; Const TempGroupList: PGroup; Var StudentsList: PStudent; Const TempStudentsList: PStudent; GroupListStringGrid: TStringGrid);

Implementation

Uses

InputEditUnit;

Function IsCorrectFileData(Const TempGroupData: TGroupData;

Const TempStudentData: TStudentData; Const IsGroup: Boolean): Boolean;

Var

BuffNumber: Integer;

BuffString: String;

IsCorrect: Boolean;

Begin

If IsGroup Then

Begin

With TempGroupData Do

Begin

IsCorrect := (Length(IntToStr(GroupNumber)) = GROUP\_LENGTH) And

(Year < 2100) And (Year > 1000) And (Length(Code) = CODE\_LENGTH)

And (CountOfStudents < MAX\_COUNT\_OF\_STUDENTS) And

(CountOfStudents > -1);

End;

End

Else

Begin

With TempStudentData Do

Begin

IsCorrect := (Length(IntToStr(GroupNumber)) = GROUP\_LENGTH) And

(Length(Name) < NAME\_LENGTH) And (Length(Surname) < NAME\_LENGTH)

And (Length(Patronymic) < NAME\_LENGTH) And (Marks[Logic] > 0) And

(Marks[Logic] < 11) And (Marks[Algebra] > 0) And

(Marks[Algebra] < 11) And (Marks[History] > 0) And

(Marks[History] < 11) And (Marks[DM] > 0) And (Marks[DM] < 11) And

(Marks[BAAP] > 0) And (Marks[BAAP] < 11) And (Marks[BPE] > 0) And

(Marks[BPE] < 11) And (AvgMark > 0) And (AvgMark < 11);

End;

End;

IsCorrectFileData := IsCorrect;

End;

Function ReadFileData(Var InputGroupFile: TGroupDataFile;

Var InputStudentsFile: TStudentDataFile; Var TempGroupList: PGroup;

Var TempStudentList: PStudent; IsGroup: Boolean): Boolean;

Var

IsCorrect: Boolean;

GroupCount: Integer;

TempGroupData: TGroupData;

TempStudentData: TStudentData;

Begin

If IsGroup Then

Begin

Reset(InputGroupFile);

IsCorrect := True;

GroupCount := 0;

While IsCorrect And (GroupCount < MAX\_COUNT\_OF\_GROUPS) And

Not EOF(InputGroupFile) Do

Begin

Read(InputGroupFile, TempGroupData);

IsCorrect := IsCorrectFileData(TempGroupData, TempStudentData, True)

And Not IsExistGroup(TempGroupList, TempGroupData);

If IsCorrect Then

Begin

AddGroup(TempGroupList, TempGroupData);

Inc(GroupCount);

End;

End;

If IsCorrect Then

IsCorrect := GroupCount <= MAX\_COUNT\_OF\_GROUPS;

CloseFile(InputGroupFile);

End

Else

Begin

Reset(InputStudentsFile);

IsCorrect := True;

While IsCorrect And Not EOF(InputStudentsFile) Do

Begin

Read(InputStudentsFile, TempStudentData);

IsCorrect := IsCorrectFileData(TempGroupData, TempStudentData,

False) And Not IsExistStudent(TempStudentList, TempStudentData);

If IsCorrect Then

Begin

AddStudent(TempStudentList, TempStudentData);

// AddStudentToGroup(TempGroupList, TempStudentData.GroupNumber);

End;

End;

CloseFile(InputStudentsFile);

End;

ReadFileData := IsCorrect;

End;

Procedure RecordFileData(Var GroupList: PGroup; Const TempGroupList: PGroup; Var StudentsList: PStudent; Const TempStudentsList: PStudent; GroupListStringGrid: TStringGrid);

Begin

StudentsList := TempStudentsList;

GroupList := TempGroupList;

GroupListStringGrid.RowCount := CountGroups(GroupList) + 1;

DrawGroupGrid(GroupListStringGrid, GroupList);

End;

Procedure WriteFileData(Var OutputGroupFile: TGroupDataFile;

Var OutputStudentFile: TStudentDataFile; Const GroupList: PGroup;

Const StudentList: PStudent; IsGroup: Boolean);

Var

CurrGroup: PGroup;

CurrStudent: PStudent;

Begin

If IsGroup Then

Begin

Rewrite(OutputGroupFile);

CurrGroup := GroupList;

While CurrGroup <> Nil Do

Begin

Write(OutputGroupFile, CurrGroup^.Data);

CurrGroup := CurrGroup^.Next;

End;

CloseFile(OutputGroupFile);

End

Else

Begin

Rewrite(OutputStudentFile);

CurrStudent := StudentList;

While CurrStudent <> Nil Do

Begin

Write(OutputStudentFile, CurrStudent^.Data);

CurrStudent := CurrStudent^.Next;

End;

CloseFile(OutputStudentFile);

End;

End;

End.