

**Федеральное государственное образовательное бюджетное учреждение
высшего образования**

«Финансовый университет при Правительстве Российской Федерации»

КОЛЛЕДЖ ИНФОРМАТИКИ И ПРОГРАММИРОВАНИЯ

ПМ.01 Разработка программных
модулей программного обеспечения
для компьютерных систем

Группа: ЗПКС-115

УТВЕРЖДАЮ

**Председатель цикловой комиссии
программирования и баз данных**

() Пестов А.И.

____.____. 2018

ПРОЕКТ КУРСОВОЙ

**На тему: Разработка автоматизированной информационной системы «Абитуриент
колледжа» на языке программирования C++**

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Руководитель курсового проекта

(Володин С. М.) _____

Исполнитель курсового проекта

(Деменчук Г.М.) _____

Оценка за проект: _____

____.____.2018

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	4
ОСНОВНАЯ ЧАСТЬ	7
1. Общий раздел	7
1.1. Системные требования	7
1.2 Характеристика системы программирования	7
1.2.1 Анализ языка программирования.....	7
1.2.2 Анализ среды реализации	17
1.3 Требования к ПО	20
1.3.1. Требования к функциональным характеристикам	20
1.3.2. Требования к надежности	20
1.3.3. Условия эксплуатации и требования к составу и параметрам технических средств	21
1.3.4. Требования к информационной и программной совместимости...	21
1.3.5. Требования к программной документации	22
2. Технологический раздел	23
2.1 Постановка задачи	23
2.1.1 Описательная модель задачи	23
2.1.2 Описание входной информацией	24
2.1.3 Описание выходной информацией	24
2.1.4 Логическая модель задачи	24
2.1.5 Требования к программе	26
2.2 Информационная модель программы	27
2.4 Тестирование программы.....	32
2.5 Анализ результатов тестирования.....	33
3. Руководство по использованию программы	34
3.1 Руководство программиста.....	34
3.1.1 Назначение и условия применения программы.....	34
3.1.2 Характеристики программы	34

3.1.3 Обращение к программе.....	34
3.1.4 Входные и выходные данные	43
3.1.5 Сообщения.....	43
3.2 Руководство пользователя	43
3.2.1 Назначение программы	43
3.2.2 Условия выполнения программы	43
3.2.3 Выполнение программы.....	44
3.2.4 Сообщение пользователю	55
ЗАКЛЮЧЕНИЕ	57
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	58
ПРИЛОЖЕНИЕ А.....	60
ПРИЛОЖЕНИЕ Б	64
ПРИЛОЖЕНИЕ В	118

ВВЕДЕНИЕ

Автоматизированная информационная система или АИС – это информационная система, использующая ЭВМ на этапах ввода информации, ее подготовки и выдачи, то есть является неким развитием ИС, которые занимаются поиском информации, используя прикладные программные средства.

Автоматизированные информационные системы представляют собой совокупность различных средств, предназначенных для сбора, подготовки, хранения, обработки и предоставления информации, удовлетворяющей информационные потребности пользователей. АИС объединяет следующие составляющие:

- языковые средства и правила, используемые для отбора, представления и хранения информации, для отображения картины реального мира в модель данных, для представления пользователю необходимой информации;
- информационный фонд системы;
- способы и методы организации процессов обработки информации;
- комплекс программных средств, реализующих алгоритмы преобразования информации;
- комплекс технических средств, функционирующих в системе;
- персонал, обслуживающий систему.

Основными целями автоматизации деятельности предприятия являются:

1. Сбор, обработка, хранение и представление данных о деятельности организации и внешней среде в виде, удобном для финансового и любого другого анализа, и использования при принятии управленческих решений.
2. Автоматизация выполнения бизнес операций (технологических операций), составляющих целевую деятельность предприятия.
3. Автоматизация процессов, обеспечивающих выполнение основной деятельности.

Автоматизированные информационные системы можно смело отнести к классу очень сложных систем и, как правило, не столько с большой физической размерностью, а в связи с многозначностью различных структурных отношений между компонентами системы.

Автоматизированная информационная система может быть легко определена как целый комплекс современных автоматизированных информационных технологий, которые предназначены для какого-либо информационного обслуживания. Без внесения самых современных методов управления, которые базируются на АИС, невозможно и повышение эффективности функционирования предприятий.

Современные АИС позволяют:

1. Повысить производительность работы всего персонала;
2. Улучшить качество обслуживания клиентской базы;
3. Снизить напряженность и трудоемкость труда персонала, а также минимизировать количество ошибок в его действиях.

На сегодняшний день, автоматизированная информационная система, является совокупностью технических (аппаратных), математических, телекоммуникационных, алгоритмических средств, методов описания и поиска объектов программирования и сбора и хранения информации. Если постараться классифицировать существующие области применения баз данных, а также оценить перспективы их развития в настоящее время, то можно получить примерный список наиболее распространенных классов, получивших распространение и применение во всех областях применения баз данных. Этот список будет выглядеть следующим образом:

- документографические и документальные применяются во всех базах органов власти и управления
- базы данных по промышленной, строительной и сельскохозяйственной продукции

- базы данных по экономической и конъюнктурной информации статистическая, кредитно-финансовая, внешнеторговая
- фактографические базы социальных данных, включающие сведения о населении и о социальной среде
- базы данных транспортных систем
- справочные данные для населения и учреждений энциклопедии и справочники, расписания самолетов и поездов, адреса и телефоны граждан и организаций
- ресурсные базы данных, включающие фактографическую информацию о природных ресурсах земля, вода, недра, биоресурсы, гидрометеорология, вторичные ресурсы и отходы, экологическая обстановка
- фактографические базы данных в области культуры и искусства
- лингвистические базы данных, то есть машинные словари разного типа и назначения.

Применение баз данных в туризме — один из способов увеличения эффективности использования данных организаций, работающих в туристической сфере.

Использование БД позволяет решить следующие важные задачи:

- хранение информации о клиентах — в данном случае БД являются источником информации с возможностью оперативного получения данных о параметрах клиентов и их заявках;
- информирование клиентов — в данном случае посредством БД пользователям предоставляется информация об услугах организации.

Целью курсового проекта является разработка АИС для регистрации и обработки информации об абитуриентах колледжей для увеличения эффективности и скорости работы приёмной комиссии. АИС должна иметь простой и интуитивно понятный интерфейс и ряд функций, таких как удаление, добавление, обновление записей в базе данных, а также печать отчетов и рейтингов.

ОСНОВНАЯ ЧАСТЬ

1. Общий раздел

1.1. Системные требования

Системные требования к электронной вычислительной машине:

- Операционная система: Windows 7 и более новые;
- Процессор с частотой 1.6 ГГц и выше;
- 1 Гб оперативной памяти;
- 13 Гб свободного пространства на жестком диске;
- Видеоадаптер с поддержкой DirectX 9, минимально допустимое разрешение экрана – 1024 x 768.

1.2 Характеристика системы программирования

Сколько существуют компьютеры, столько же и существуют средства разработки программного обеспечения. С течением времени они совершенствуются, постоянно прогрессируют и с каждым разом становятся все удобнее в использовании и проще в изучении.

Для начала подвергнем анализу наиболее популярные языки программирования: Java, C#, C++ и Visual Basic [Data Structures..., 1999].

1.2.1 Анализ языка программирования

Таблица 1. Парадигмы

Возможность	Язык			
	Java	C#	C++	Visual Basic
Императивная	+	+	+	+
Объектно-ориентированная	+	+	+	+
Рефлексивная	-/+	+/-	+/-	+/-
Обобщенное программирование	+	+	+	+
Логическая	-	-	-	-
Распределительная	+	-/+	+/-	-

- Императивная. Императивный язык описывает не задачу, а способ ее решения.
- Объектно-ориентированная. Она основана на представлении всего в языке в виде объектов. Объекты, в свою очередь, могут содержать в себе как переменные, так и методы для работы с ними. Выполняется поддержка основополагающих принципов ООП: полиморфизм, наследование и инкапсуляция.
- Рефлексивная. Это возможность, предполагающая, что язык программирования может оперировать собственным кодом как данными.
- Обобщенное программирование. При написании алгоритмов есть возможность передавать им данные любых типов.
- Логическая. Программа является набором некоторых правил и фактов вывода в определенном логическом исчислении.
- Распределительная. Поддержка языком программирования возможности распараллеливания вычислений.

Таблица 2. Типизация

Возможность	Язык			
	Java	C#	C++	Visual Basic
Статическая типизация	+	+	+	+
Явная типизация	+	+	+	+
Неявное приведение типов без потери данных	-	+	+	+
Неявное приведение типов с потерей данных	-	-	+	+
Неявное приведение типов в неоднозначных ситуациях	-	+	+	+
Алиасы типов	-	+	+	-
Информация о типах в runtime	+	+	-/+	+
Информация о типах-параметрах в runtime	-	+	-/+	+

- Статическая типизация. Невозможность изменения типов в процессе выполнения после их объявления.
- Явная типизация. Параметры и типы переменных указываются явно.
- Неявное приведение типов без потери данных. Например, приведение числа с плавающей точкой к целому числу.
- Неявное приведение типов с потерей данных. Например, приведение целого числа к числу с плавающей точкой.
- Неявное приведение типов в неоднозначных ситуациях. Допустим, при попытке сложения числа 7 и строки «3» мы можем получить, как число 10, так и строку «73».
- Алиасы типов. Замена одного алиаса типа на другой, полностью эквивалентный ему, например, `#typedef TYPE1 TYPE2`.
- Информация о типах в runtime. Наличие возможности выяснить точный тип объекта в runtime.

Таблица 3. Компилятор-интерпретатор

Возможность	Язык			
	Java	C#	C++	Visual Basic
Open-source	+	+	+	+
Возможность компиляции	+	+	+	+
Bootstrapping	+	+	+	?
Многопоточная компиляция	+	-	+	+
Интерпретатор командной строки	-	-	+/-	+
Условная компиляция	-/+	+	+	+

- Open-source компилятор или интерпретатор. Эта возможность говорит о наличии полноценного open-source компилятора или интерпретатора.

- Возможность компиляции. Способность компилировать код в нативный или в byte-cod с возможностью JIT компиляции.
- Bootstrapping. Это значит, что компилятор написан на том же языке, что и компилируется им.
- Многопоточная компиляция. Способность компилятора распараллеливать процесс сборки на несколько потоков при условии наличия таковых.
- Интерпретатор командной строки. Возможность вводить инструкции языка в командную строку с последующим незамедлительным их выполнением.
- Условная компиляция. Наличие возможности включения или отключения компиляции участка программного кода в зависимости от выполнения условий, например, в C++ это делается с помощью `#if ... #endif`.

Таблица 4. Управление памятью

Возможность	Язык			
	Java	C#	C++	Visual Basic
Создание объектов на стеке	-	+	+	-
Неуправляемые указатели	-	+	+	-
Ручное управление памятью	-	+	+	-
Сборка мусора	+	+	-/+	+

- Объекты на стеке. Способность создавать экземпляры любого типа данных не «в куче», а на стеке.
- Неуправляемые указатели. Характерен прямой доступ к памяти и наличие адресной арифметики.
- Ручное управление памятью. Возможность оперировать с данными «в куче» посредством, например, операторов `new` и `delete` в языке C++.

– Сборка мусора. Компилятор способен сам отслеживать неиспользуемые участки памяти в куче и освобождать их.

Таблица 5. Управление потоками вычислений

Возможность	Язык			
	Java	C#	C++	Visual Basic
Конструкция goto	-	+	+	+
Инструкция break без метки	+	+	+	+
Инструкция break с меткой	+	-	-	+
Поддержка try/catch	+	+	+	+
Ленивые вычисления	-	-/+	+	-

– Конструкция goto. Подразумевает возможность безусловного перехода к метке.

– Инструкция break без метки. Безусловный выход из ближайшего цикла.

– Инструкция break с меткой. Поддержка безусловного выхода из цикла, помеченного меткой.

– Поддержка try/catch. Наличие возможности обрабатывать исключения с помощью конструкции try/catch.

– Ленивые вычисления. Предполагает экономию времени на проведении вычислений, результаты которых в дальнейшем не понадобятся.

Таблица 6. Типы и структуры данных

Возможность	Язык			
	Java	C#	C++	Visual Basic
Кортежи	-	+/-	+/-	+/-
Многомерные массивы	+/-	+	+	+
Динамические массивы	+/-	+/-	+	+
Ассоциативные массивы	+/-	+	+	+
Контроль границ массивов	+	+	+/-	+
Цикл <code>foreach</code>	+	+	+	+
Целые числа произвольной длины	+	+	-	+
Целые числа с контролем границ	-	-	-	-

- Многомерные массивы. Наличие в языке возможности создавать многомерные массивы, например, `array[N][M]`.
- Динамические массивы. Наличие в языке возможности создавать массивы, которые способны изменять свой размер в процессе выполнения программного кода.
- Ассоциативные массивы. Это так называемые hash-таблицы.
- Цикл `foreach`. Наличие в языке конструкции, благодаря которой существует возможность перебора всех элементов коллекции.
- Наличие поддержки длинной целочисленной арифметики.
- Целые числа с контролем границ. Определение типа чисел с заданным диапазоном, например, `int range [-77, 69]`, и при попытке присвоения переменной `range` значения, выходящего за границы, происходила бы ошибка.
- Кортежи. Способность компилятора вернуть из метода кортеж. Кортеж – это неименованный тип данных, содержащий безымянные поля произвольного типа.

Таблица 7. Объектно-ориентированные возможности

Возможность	Язык			
	Java	C#	C++	Visual Basic
Интерфейсы	+	+	+	+
Мультиметоды	-	-/+	-/+	-
Переименование членов при наследовании	-	-	-/+	-
Множественное наследование	-	-	+	-

– Интерфейсы. Наличие синтаксической и семантической конструкций в программном коде, обеспечивающая специфицирование услуг, предоставляемых классом.

– Множественное наследование. Способность при создании класса наследоваться от нескольких, что позволяет производному классу содержать в себе функционал базовых классов.

Таблица 8. Функциональные возможности

Возможность	Язык			
	Java	C#	C++	Visual Basic
First class function	-	+	+	?
Анонимные функции	-	+	+	+
Лексические замыкания	+	+	+	+
Частичное применение	-	?	+/-	?
Каррирование	-	+	+/-	-

– First class function или объекты первого класса. В контексте определенного языка это названия сущностей, способных передаваться в качестве параметра, быть полученными при выполнении функции или быть присвоенными переменной.

– Анонимные функции. Наличие особого вида функций, объявляемых в месте использования и не получающих уникального идентификатора для их вызова.

– Лексическое замыкание. Наличие в функции ссылки на переменную, объявленную вне тела функции и не переданную в нее в качестве параметра.

Таблица 9. Прочие свойства

Возможность	Язык			
	Java	C#	C++	Visual Basic
Макросы	-	-	+	+
Шаблоны	+	+	+	+
Поддержка Unicode в идентификаторах	+	+	+	+
Перегрузка функций	+	+	+	+
Динамические переменные	?	?	+	+
Значение параметров по умолчанию	-	+	+	+
Локальные функции	+/-	+/-	+	+/-
Наличие библиотек для работы с графикой	+	+	+	+

– Макросы. Наличие макросистемы, обрабатывающей код программы до ее компиляции или выполнения.

– Шаблоны. Наличие возможности создания обобщенных классов для расширения функционала. Например, в C++ это template классы.

– Перегрузка функций. Возможность создания перегруженных функций, позволяющих принимать на вход различные наборы параметров.

– Динамические переменные. Способность языка создавать в нем переменные «в куче».

– Значение параметров по умолчанию. Способность вызывать конструкторы или функции без явной подстановки значений входных параметров, которые проинициализированы в описании функции.

– Наличие библиотек для работы с графикой. Возможность использовать функционал таких библиотек как: OpenGL, WebGL, OpenML, DirectX.

Исходя из приведенных сравнительных характеристик можно сказать, что по своей универсальности при написании программного обеспечения языку C++ нету равных. Считается, что при реализации программного продукта нужно отталкиваться от поставленной цели, и для этого в наибольшей степени подходят два языка: C++ и Java. Они оба обладают возможностью распараллеливания вычислений и созданием хорошего графического интерфейса, хотя в Java для этого требуется значительно больше оперативной памяти, что склоняет выбор не в его пользу. Также, в отличие от языка C++, Java не поддерживает прямой доступ к памяти компьютера и ленивые вычисления, что сильно увеличило бы объем кода. Помимо прочего C++ обладает более хорошей типизацией данных.

Язык программирования C++ задумывался как язык, который будет:

- лучше языка C
- поддерживать абстракцию данных;
- поддерживать объектно-ориентированное программирование

C++ - язык общего назначения. За исключением второстепенных деталей он содержит язык C как подмножество. Язык C расширяется введением более гибких и эффективных средств, предназначенных для построения новых типов. Программист структурирует свою задачу, определив новые типы, которые точно соответствуют понятиям предметной области задачи. Такой метод построения программы обычно называют абстракцией данных. Информация о типах содержится в некоторых объектах типов, определенных пользователем. С такими объектами можно работать надежно и просто даже в тех случаях, когда их тип нельзя установить на стадии трансляции. Программирование с использованием таких объектов обычно называют объектно-ориентированным. Если этот метод применяется правильно, то программы становятся короче и понятнее, а сопровождение их упрощается.

Ключевым понятием C++ является класс. Класс - это определяемый пользователем тип. Классы обеспечивают целостностью данных, их инициализацию, неявное преобразование пользовательских типов, динамическое задание типов, контролируемое пользователем управление памятью и средства для перегрузки операций. В языке C++ концепции контроля типов и модульного построения программ реализованы более полно, чем в C. Кроме того, C++ содержит усовершенствования, прямо с классами не связанные: символические константы, функции-подстановки, стандартные значения параметров функций, перегрузка имен функций, операции управления свободной памятью и ссылочный тип. В C++ сохранены все возможности C эффективной работы с основными объектами, отражающими аппаратную "реальность" (разряды, байты, слова, адреса и т.д.). Это позволяет достаточно эффективно реализовывать пользовательские типы.

Объектно-ориентированное программирование - это метод программирования, способ написания "хороших" программ для множества задач. Если этот термин имеет какой-то смысл, то он должен подразумевать: такой язык программирования, который предоставляет хорошие возможности для объектно-ориентированного стиля программирования.

Язык C++ проектировался для поддержки абстракции данных и объектно-ориентированного программирования в дополнение к традиционному стилю C., впрочем, это не значит, что язык требует какого-то одного стиля программирования от всех пользователей.

Развитие языка C++ происходило на базе языка C, и, за небольшим исключением, C был сохранен в качестве подмножества C++. Базовый язык C был спроектирован таким образом, что имеется очень тесная связь между типами, операциями, операторами и объектами, с которыми непосредственно работает машина, т.е. числами, символами и адресами. За исключением операций new, delete и throw, а также проверяемого блока, для выполнения

операторов и выражений C++ не требуется скрытой динамической аппаратной или программной поддержки.

Первоначально язык C задумывался как конкурент ассемблера, способный вытеснить его из основных и наиболее требовательных к ресурсам задач системного программирования. В проекте C++ были приняты меры, чтобы успехи C в этой области не оказались под угрозой. Различие между двумя языками прежде все состоит в степени внимания, уделяемого типам и структурам. Язык C выразителен и в то же время снисходителен по отношению к типам. Язык C++ еще более выразителен, но такой выразительности можно достичь лишь тогда, когда типам уделяют большое внимание. Когда типы объектов известны, транслятор правильно распознает такие выражения, в которых иначе программисту пришлось бы записывать операции с утомительными подробностями. Кроме того, знание типов позволяет транслятору обнаруживать такие ошибки, которые в противном случае были бы выявлены только при тестировании. Отметим, что само по себе использование строгой типизации языка для контроля параметров функции, защиты данных от незаконного доступа, определения новых типов и операций не влечет дополнительных расходов памяти и увеличения времени выполнения программы.

1.2.2 Анализ среды реализации

Visual Studio 2017 (VS) представляет собой интегрированную среду разработки (Integrated Development Environment, IDE). IDE — это набор инструментов разработчика ПО, собранный в составе единого приложения и облегчающий труд программиста при написании приложений. Без IDE (в данном случае — без VS) для написания программы требуется текстовый редактор, с помощью которого программист вводит весь исходный код своей будущей программы. Затем, когда исходный код написан, необходимо запустить из командной строки компилятор, чтобы создать исполняемый файл приложения. Основная проблема, связанная с использованием текстового

редактора и компилятора, запускаемого из командной строки, заключается в том, что вы выполняете большое количество ручной работы и теряете при этом много времени. К счастью, с помощью VS многие из этих рутинных и трудоемких задач, связанных с повседневной работой программиста, можно автоматизировать.

В состав VS входит целый набор типовых проектов, из которых каждый разработчик может подобрать именно то, что ему в данный момент требуется. Каждый раз, когда разработчик создает новый проект, VS автоматически моделирует "скелет" будущего приложения, причем этот код можно немедленно скомпилировать и запустить на исполнение. В составе каждого типового проекта имеются элементы, которые по желанию добавлять в ваш проект. Любой проект, в любом случае, содержит автоматически сгенерированный код, который представляет собой основу будущей программы. VS предлагает множество готовых к использованию элементов управления, включая и код, необходимый для их создания. Это экономит время разработчиков, избавляя их от необходимости каждый раз заново создавать типовой программный код для решения часто встречающихся задач. Многие из более сложных элементов управления содержат так называемые "программы-мастера" (Wizards), которые помогают настроить поведение элементов управления, автоматически генерируя код в зависимости от выбранных вами опций.

Редактор VS оптимизирует работу программиста по кодированию. Существенная часть синтаксических элементов новой программы выделяется при помощи системы цветовых обозначений. В вашем распоряжении окажутся такие технологии, как Intellisense, известная как автодополнение. В ходе того, как вы будете вводить новый код, на экране будут появляться всплывающие подсказки. Наконец, для ускорения выполнения многих задач вам будет предоставлено большое количество клавиатурных комбинаций (keyboard shortcuts). Существует и набор средств быстрой переработки или рефакторинга

(refactoring), которые позволяют быстро усовершенствовать структуру кода, не отрываясь от процесса программирования. Например, функция переименования (Rename) позволяет изменить имя идентификатора там, где оно определено, и это повлечет за собой изменение имени данного идентификатора повсюду, где он встречается в коде программы. Кроме того, VS предоставляет и множество других, не менее удобных возможностей. Например, иерархия вызовов (callhierarchy) позволяет программисту проследить всю цепочку вызовов в коде приложения, с верхних уровней до самых нижних. Еще одна возможность, автоматически генерируемые фрагменты (snippets), позволяет вводить сокращения, которые разворачиваются в шаблоны кода (code template). Наконец, списки действий (action lists) предназначены для автоматической генерации нового кода.

Вам действительно необходимо научиться ориентироваться в среде VS, чтобы разобраться во всем множестве новых возможностей, призванных упростить нелегкую задачу быстрой разработки высококачественных приложений. Так, окно Toolbox просто "до отказа" забито различными элементами управления, окно Server Explorer предназначено для работы с сервисами и базами данных операционной системы, а окно Solution Explorer дает возможность работать с вашими проектами, тестировать утилиты, и предоставляет средства визуального проектирования. Кроме того, предоставляются и средства работы с компиляторами.

Многие из элементов, образующих среду VS, являются настраиваемыми. Это значит, что вы можете менять цвета отображения элементов кода, опции редактора, а также общее оформление. Набор опций настолько обширен, что и в самом деле необходимо потратить некоторое время, чтобы ознакомиться с ними и привыкнуть к тому, где и какую из них следует искать. Если стандартные настройки среды VS, которые по умолчанию предлагаются сразу же после установки, вас не устраивают, или не предлагают нужных вам опций, вы можете написать собственные макросы, чтобы автоматизировать

последовательности часто выполняемых шагов. Для более сложных вариантов настройки VS предлагает специальный интерфейс прикладного программирования (Application Programming Interface, API), предназначенный для создания собственных дополнительных модулей (add-ins) и расширений (extensions). Некоторые сторонние разработчики уже предлагают приложения, способные интегрироваться в среду VS. В качестве примера можно назвать среду разработки на Delphi от компании Embarcadero¹, которая встраивается в Visual Studio.

1.3 Требования к ПО

1.3.1. Требования к функциональным характеристикам

Программа должна осуществлять добавление, редактирование, удаление, просмотр информации, а также ограничить доступ к данным посторонним лицам.

Программа должна:

- Работать с заданным алгоритмом функционирования;
- Поддерживать диалоговый режим в рамках предоставляемых пользователю возможностей;
- Производить бесперебойную работу по преобразованию информации.

1.3.2. Требования к надежности

- состав и количественные значения показателей надежности для системы в целом или ее подсистем;
- перечень аварийных ситуаций, по которым должны быть регламентированы требования к надежности, и значения соответствующих показателей;
- требования к надежности технических средств и программного обеспечения;

- требования к методам оценки и контроля показателей надежности на разных стадиях создания системы в соответствии с действующими нормативно-техническими документами.

1.3.3. Условия эксплуатации и требования к составу и параметрам технических средств

Условия эксплуатации программы совпадают с условиями эксплуатации по ЭВМ IBM PC. Программа должна быть рассчитана на непрофессионального пользователя.

Минимальные требования к электронной вычислительной машине:

- Процессор: Intel Celeron или аналогичный AMD;
- Частота: 1500 МГц;
- Оперативная память: 512 Мб;
- Клавиатура и мышь.

Рекомендуемые требования к электронной вычислительной машине:

- Процессор: Intel Pentium;
- Частота: 2000 МГц;
- Оперативная память: 1 Гб;
- Клавиатура и мышь.

1.3.4. Требования к информационной и программной совместимости

Операционная Система: windows 7; 8; 10;

Язык программирования: C++

Требования к транспортировке и хранению:

Программа поставляется в электронном виде, а также на диске.

Программная документация поставляется в электронном и печатном виде.

Специальные требования:

Программное обеспечение должно иметь простой, интуитивно понятный интерфейс, рассчитанный на непрофессионального пользователя.

Документация на принятое эксплуатационное программное обеспечение (ПО) должна содержать полную информацию, необходимую для работы программистов с данной программой.

1.3.5. Требования к программной документации

Основными документами, регламентирующими разработку будущих программ, должны быть документы Единой Системы Программной Документации (ЕСПД): руководство системного программиста, руководство оператору, описание программы.

Основная часть пояснительной записки курсового проекта должна отражать все этапы работы студента для достижения поставленной цели, а также последовательное решение всех поставленных им задач.

2. Технологический раздел

2.1 Постановка задачи

2.1.1 Описательная модель задачи

Для оформления документов в колледж абитуриенту необходимо сдать относительно небольшое кол-во документов: аттестат за 9 или 11 класс и портфолио, но т.к. абитуриентов достаточно много, а людей в приемной комиссии, наоборот, очень мало, то данные мероприятия могут занимать достаточно продолжительное количество времени, которое можно потратить на более рациональные вещи.

Существует два варианта решения данной задачи:

1. Прием новых сотрудников для приемной комиссии
2. Оптимизация процесса приема документов в приемной комиссии

Прием новых сотрудников не особо рациональное действие: каждому следует каждый месяц выплачивать зарплату и размещать большее количество рабочих мест (стол, стул, бумага и прочие расходные материалы), поэтому мы будем действовать по второму пункту.

Оптимизировать процесс приема документов рациональнее всего с помощью автоматизации каких-либо базовых действий: занесение информации о среднем балле аттестата, формирование рейтинга абитуриентов или выставление текущей даты приема – все это можно реализовать с помощью автоматизированной системы приема абитуриентов.

Достижение цели разбивается на ряд задач. В процессе достижения основной цели создания АИС решаются следующие основные задачи:

1. Информатизация регистрации данных об абитуриентах: Предполагает ведение базы данных с несколькими таблицами, содержащими информацию об абитуриентах колледжа. На основании

полученных данных ведется формирование рейтинга абитуриентов с определенной выборкой по направлению (ПКС/ИБАС) или по типу аттестата (оригинал/копия)

2. Редактирование базы данных — удаление, обновление, добавление данных абитуриентов.

3. Формирование отчетов. Создание отчетов по каждому абитуриенту при приеме документов в колледж, а также при формировании рейтинга абитуриентов с их последующим выводом на печать.

2.1.2 Описание входной информацией

Входные данные — содержание элементов WindowsForms, как: TextBox, Combobox и RadioButton. Более подробное описание входной информации представлено в приложении А.

2.1.3 Описание выходной информацией

Выходные данные — база данных SQLite3, документы Microsoft Word, а также содержание элементов WindowsForms GridTable. Более подробное описание выходной информации представлено в приложении А.

2.1.4 Логическая модель задачи

Логическая модель задачи приведена на рисунке 2.1. Она демонстрирует последовательность выполнения действий, реализующих принцип функционирования программы.

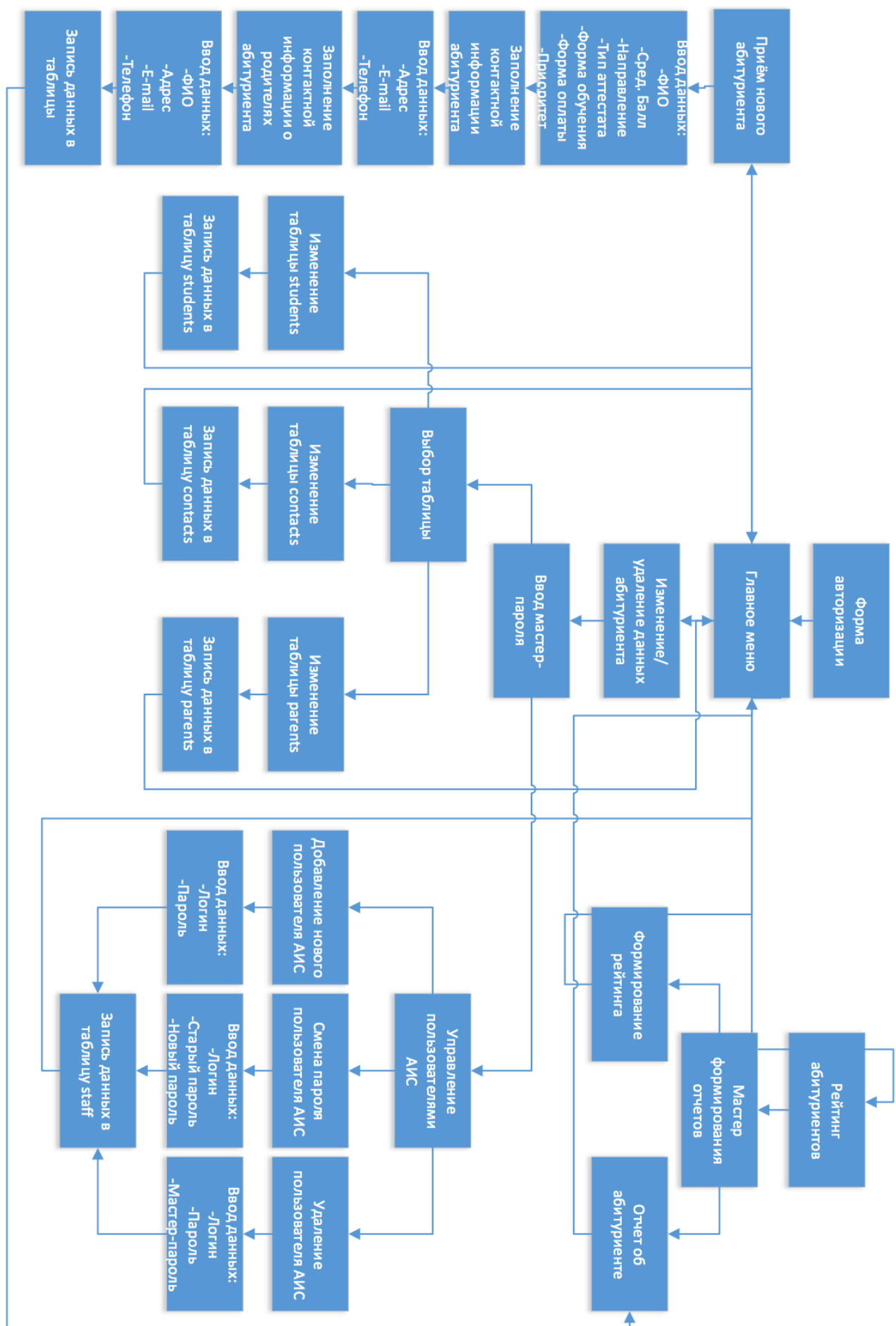


Рисунок 2.1 - Логическая модель задачи

2.1.5 Требования к программе

Система должна сохранять работоспособность и обеспечивать восстановление своих функций при возникновении следующих внештатных ситуаций:

- при сбоях в системе электроснабжения аппаратной части, приводящих к перезагрузке ОС, восстановление программы должно происходить после перезапуска ОС и запуска исполняемого файла системы;
- при ошибках в работе аппаратных средств (кроме носителей данных и программ) восстановление функции системы возлагается на ОС;
- при ошибках, связанных с программным обеспечением (ОС и драйверы устройств), восстановление работоспособности возлагается на ОС.

В целях надежности системы она должна удовлетворять следующим требованиям:

- разработанная программа должна обладать средствами защиты от ошибочных действий пользователей;
- все ошибки должны отображаться с комментариями или подсказками по их устранению;
- гарантировать сохранность данных при сбоях в работе внешних устройств.

Для повышения надежности необходимо принять следующие меры:

- сконфигурировать аппаратные и программные средства в соответствии с техническими требованиями;
- периодически осуществлять резервное копирование информации;
- регулярно проверять целостность базы данных;
- поддерживать исправность оборудования.

2.2 Информационная модель программы

На рисунке 2.2 приведена информационная модель программы, состоящая из 3 таблиц: students, contacts и parents. Разберем каждую из них подробнее.

Таблица students – основная информация об абитуриенте колледжа:

- ФИО
- Средний балл
- Приоритет зачисления
- Форма обучения
- Специальность
- Тип аттестата
- Форма оплаты
- Дата подачи заявления

Таблица contacts – контакты абитуриента, дополнительная информация:

- Номер студента (для связи таблиц)
- Адрес
- E-mail
- Телефон

Таблица parents – контакты родителей (представителей) абитуриента:

- Номер студента (для связи таблиц)
- ФИО
- Адрес
- E-mail
- Телефон

Таблица staff – данные сотрудников приемной комиссии колледжа в виде зашифрованного MD5 хеша:

- Номер
- Логин
- Пароль



Рисунок 2.2 - Информационная модель программы

2.3 Логическая модель программы

Логическая модель программы приведена на рисунке 2.3. Она демонстрирует последовательность выполнения действий, реализующих принцип функционирования программы. Последовательность работы программы:

1. Ввод логина и пароля сотрудника приемной комиссии, переход к пункту 2 в случае успешной авторизации.

2. Выбор функций, главное меню программы.

- 2.1. Прием нового абитуриента. Пользователь вводит основную информацию об абитуриенте, а именно: ФИО, средний балл, направление, тип аттестата, форма обучения, форма оплаты, а также приоритет по конкурсу. Запись данных в таблицу `students`. Предложение о вводе контактной информации об абитуриенте.

- 2.1.1. Заполнение контактной информации об абитуриенте: телефон, адрес проживания и электронная почта (e-mail). Запись данных в таблицу `contacts`. Предложение о вводе контактной информации родителей абитуриента.

- 2.1.2. Заполнение контактной информации родителей абитуриента: ФИО, e-mail, адрес проживания и телефон. Запись данных в таблицу `parents`.

Предложение о формировании отчета о приеме документов в колледж.

- 2.1.3. Формирование отчета. Возможность выбора данных для формирования отчета: основная информация абитуриента, контактные данные абитуриента, контактные данные родителей абитуриента, дата формирования отчета, время формирования отчета и поле для подписи. После формирования отчета в Microsoft Word происходит переход в главное меню программы (пункт 2).

- 2.2. Изменение/удаление данных абитуриента.

- 2.2.1. Запрос пароля на изменение данных абитуриентов. При нажатии кнопки «Далее» происходит сравнение MD5 хеша введенной строки с хешем, заранее записанным в глобальном статическом классе. Если пароль

верный, то происходит переход к следующему пункту, иначе поступает предложение ввести пароль еще раз.

2.2.2. Форма выбора таблицы для обновления, удаления или добавления данных, также возможен доступ к форме управления пользователями АИС. Пользователь выбирает одну из трех таблиц (students, contacts или parents) для дальнейшей работы с ней или переходит на форму управления пользователями АИС.

2.2.3. Форма взаимодействия с таблицей. Вывод выбранной таблицы на экран со всеми ее полями. При добавлении/удалении, вводимые данные в Gridtable проходят проверку на ввод перед добавлением в какую-либо таблицу базы данных. После окончания работы с таблицей происходит переход в главное меню программы (пункт 2).

2.2.4 Форма управления пользователями (сотрудниками приемной комиссии). Возможны следующие действия: Добавление нового пользователя, смена пароля пользователя, а также удаление пользователя АИС. Изменения вносятся в таблицу.

2.3 Рейтинг абитуриентов. Просмотр таблицы students в наглядном виде, возможна сортировка по любым полям таблицы (№ заявления, ФИО, средний балл, приоритет, форма обучения, специальность, тип аттестата, форма оплаты, дата приема документов). По желанию пользователя возможен запуск формы выбора отчетов.

2.3.1 Форма выбора отчетов. Пользователю предоставлена возможность выбора 2 типов отчетов с параметрами: формирование рейтинга, а также отчет о конкретном абитуриенте.

2.3.2 Форма рейтинга абитуриентов. Через элемент Gridtable выводится таблица students, возможна сортировка по любому столбцу. Также возможен переход к мастеру формирования отчетов. В мастере формирования отчетов возможен выбор 2 типов отчетов: формирование рейтинга с выборкой по специальности и форме оплаты, а также отчет о конкретном абитуриенте.

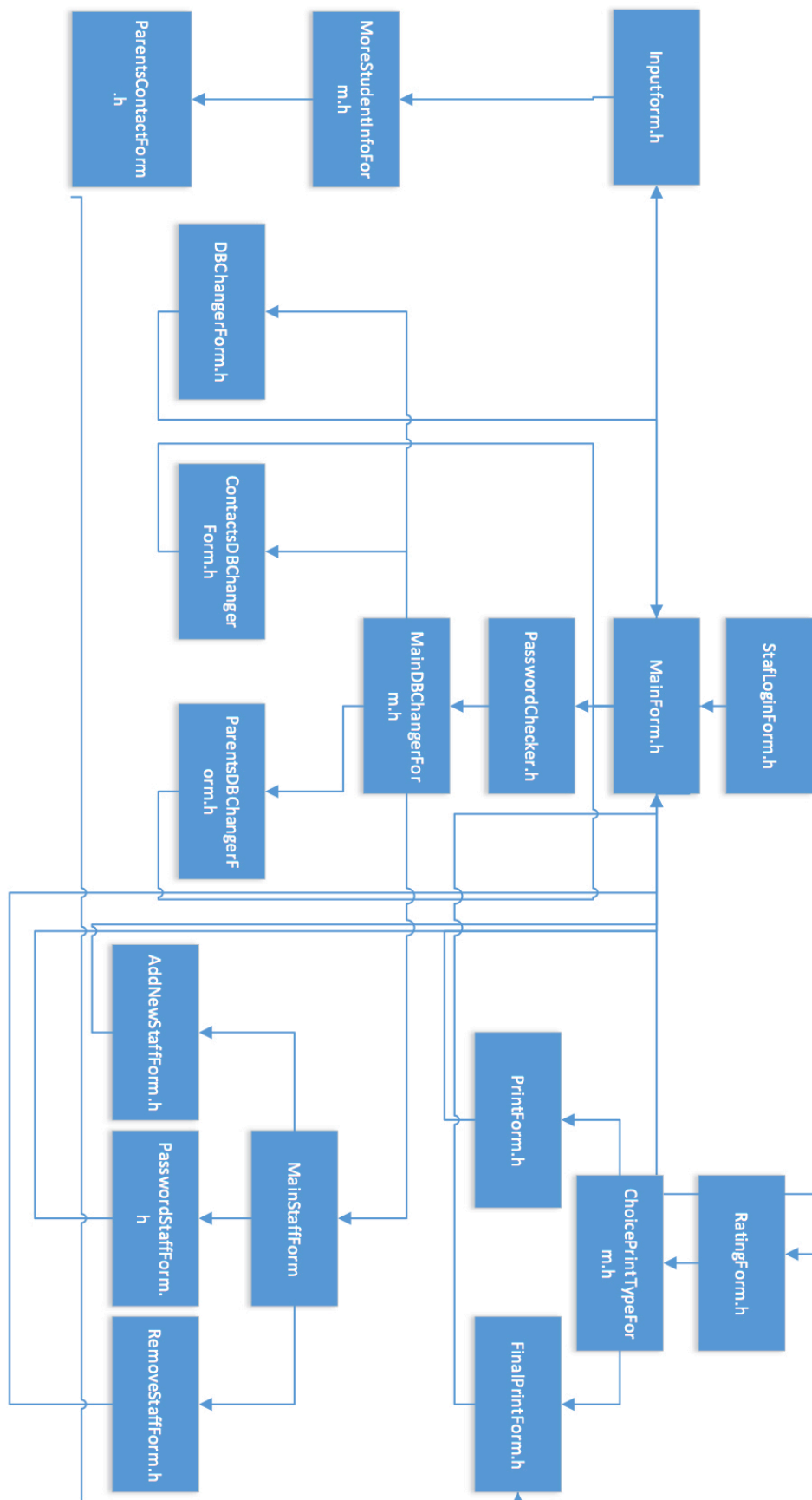


Рисунок 2.3 – Логическая модель программы

2.4 Тестирование программы

Проверка на некорректный ввод осуществляется по каждому элементу формы: ФИО, средний балл, направление, а также все RadioButton (рисунок 2.4). При неправильной функции валидации невозможно перейти на форму заполнения дополнительных данных абитуриента т.к. выводится соответствующее сообщение (рисунок 2.5). Аналогично происходит взаимодействие с остальными формами.

Надежность программы рассчитываем по формуле $\frac{a}{b}$,

где a – количество успешных запусков;

b – общее количество запусков;

Надежность программы $\frac{35}{36} \approx 0,972 = 97\%$, что удовлетворяет требованиям, выдвигаемым к ней.

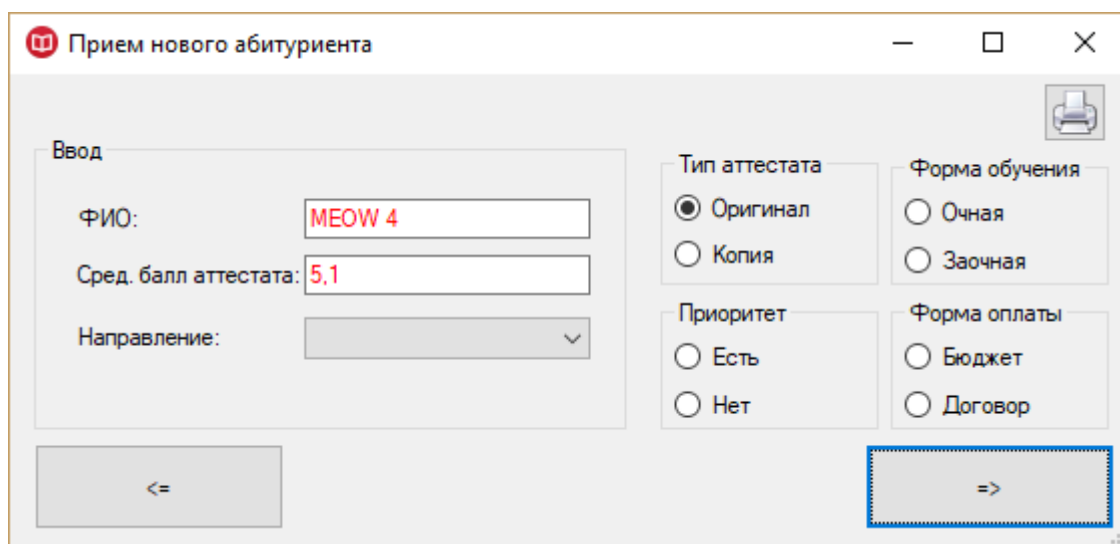


Рисунок 2.4 – Пример некорректного ввода

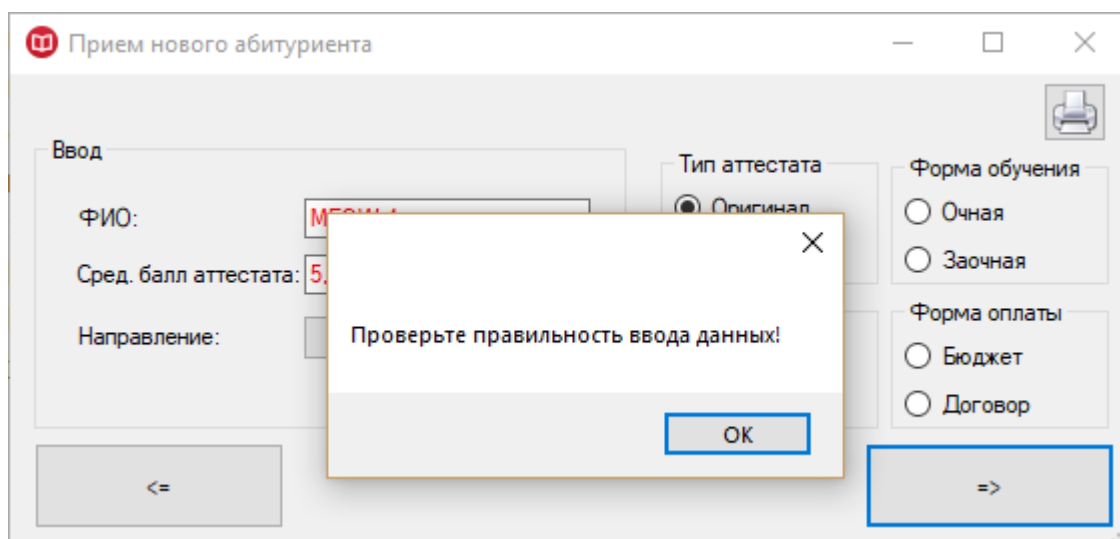


Рисунок 2.5 – Сообщение о некорректном вводе

2.5 Анализ результатов тестирования

Из предыдущих примеров в разделе выше видно, что программа выполняет все свои функциональные требования, а именно:

- Последовательно записывает введенные данные с фильтрацией по инициативе пользователя;
- Формирует отчеты о абитуриенте и об общем рейтинге абитуриентов с определенной выборкой по выбору пользователя (форма оплаты и специальность);
- Успешно добавляет и обновляет данные с фильтрацией из формы изменения таблиц (students, parents, contacts);
- Отрабатывает действия с администрированием пользователей АИС: добавление, смена пароля, удаление.

3. Руководство по использованию программы

3.1 Руководство программиста

3.1.1 Назначение и условия применения программы

Программа предназначена для упрощения зачисления абитуриента в среднее профессиональное образовательное учреждение. Для запуска должны быть соблюдены минимальные системные требования (см пункт 1.1).

Для использования этого ПО необходим персональный компьютер с установленной ОС Windows не ниже Windows 7, мышь, клавиатура, монитор, Microsoft Office Word для формирования отчетов.

3.1.2 Характеристики программы

Программа разрабатывалась для платформы Windows. Запуск на альтернативных платформах возможен только с использованием виртуализации/виртуальных машин.

Данная программа представляет собой графическое приложение с 18 формами. Состоит из исполняемого файла «ДЕМКА.exe» и файла базы данных SQLite3 «database_vs.db», содержащим таблица students, contacts, parebts и staff (см пункт 2.2)

Также возможно прямое открытие .db файла с помощью программы «DB Browser for SQLite» без использования АИС. В целях безопасности пары логинов/паролей зашифрованы согласно алгоритму (см. пункт 3.1.3 рисунки 3.6 – 3.7).

3.1.3 Обращение к программе

При первом запуске программы Вам требуется войти с парой логина/пароля (рисунок 3.1). Все данные конфиденциальны и зашифрованы с помощью хеша MD5.

При правильной паре логина и пароля выводится сообщение об успешной авторизации (рисунок 3.2), в противном случае выводится сообщение о неверном пароле (рисунок 3.3).

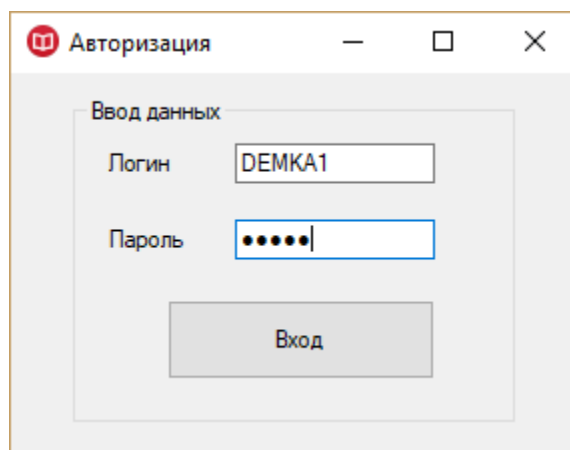
The image shows a window titled 'Авторизация' (Authorization) with a red icon. Inside, there's a 'Ввод данных' (Data input) section. It contains two text boxes: 'Логин' (Login) with the text 'ДЕМКА1' and 'Пароль' (Password) with five dots. Below these is a 'Вход' (Login) button.

Рисунок 3.1 – Форма авторизации пользователя

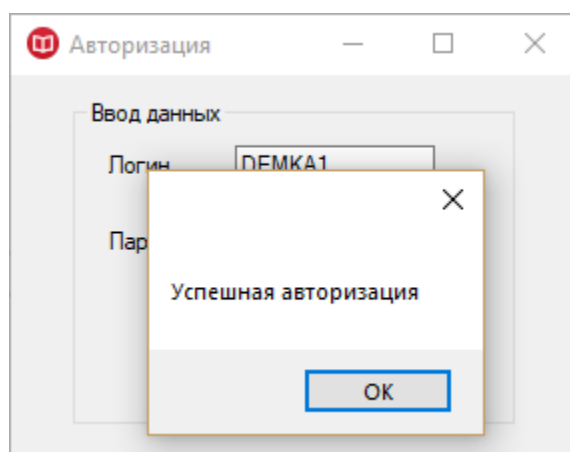
The image shows the same 'Авторизация' window as in Figure 3.1, but with a modal dialog box on top. The dialog box has a title bar with a close button and contains the text 'Успешная авторизация' (Successful authorization) and an 'ОК' button.

Рисунок 3.2 – Сообщение об успешной авторизации пользователя

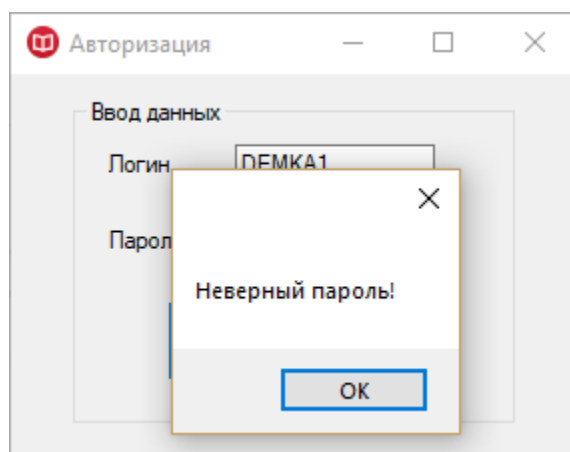
The image shows the same 'Авторизация' window as in Figure 3.1, but with a modal dialog box on top. The dialog box has a title bar with a close button and contains the text 'Неверный пароль!' (Incorrect password!) and an 'ОК' button.

Рисунок 3.3 – Сообщение о неправильной паре логина/пароля

В случае успешной авторизации вам будет доступна главная форма действий с АИС (рисунок 3.4). Для конфигурации программы администратору необходимо перейти на форму изменения/удаления данных абитуриента. При нажатии пользователя на форму изменения/удаления данных абитуриента программа запрашивает мастер-пароль администратора АИС (рисунок 3.5) и сравнивает с зашифрованной записью в таблице staff БД (рисунки 3.6 – 3.7), согласно алгоритму. Мастер-пароль невозможно сменить каким-либо способом. Сброс мастер-пароля возможен лишь при полном удалении .db файла базы данных SQLite3 с последующей утратой всех данных, находящихся в ней.

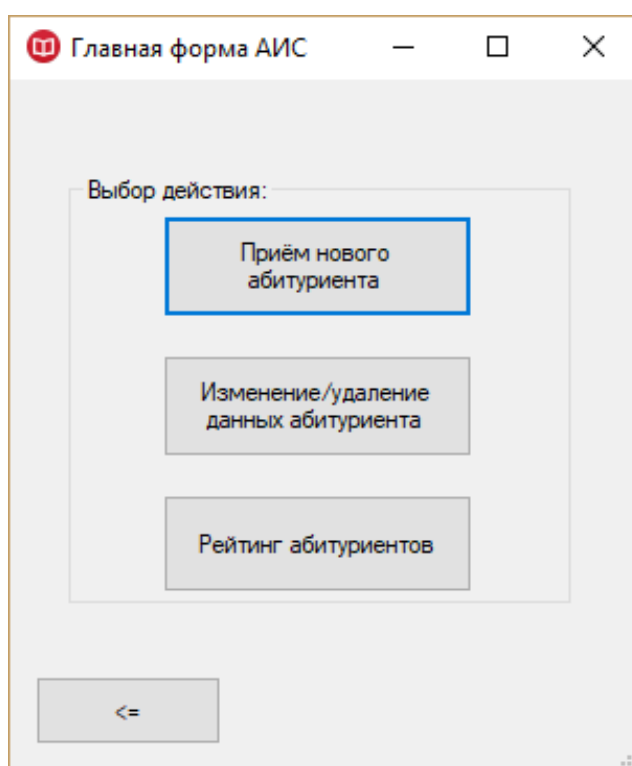


Рисунок 3.4 – Главная форма АИС

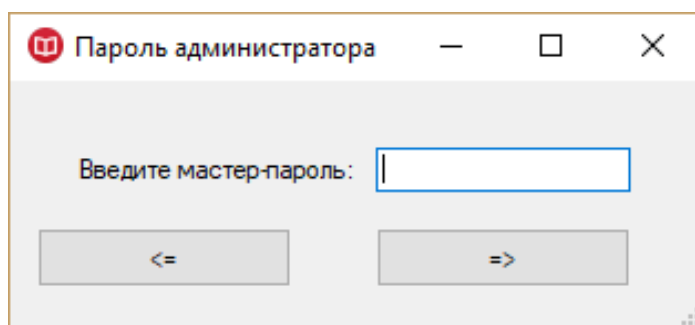


Рисунок 3.5 – Запрос мастер-пароля

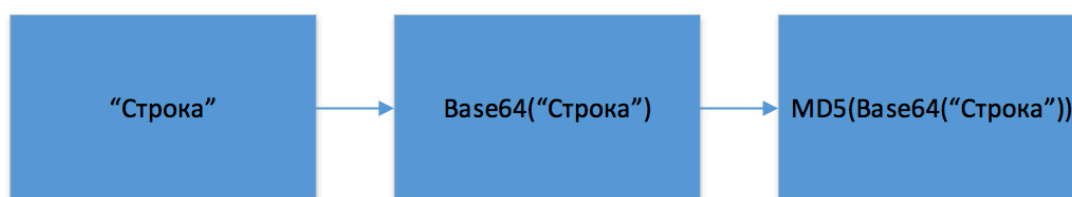


Рисунок 3.6 – Последовательность шифрования пароля и логина



Рисунок 3.7 – Пример шифрования логина

После успешной авторизации администратор попадает в меню действий (рисунок 3.8). В данном меню возможны любые действия со всеми тремя таблицами (students, contacts, parents), а также управление (администрирование) пользователями АИС.

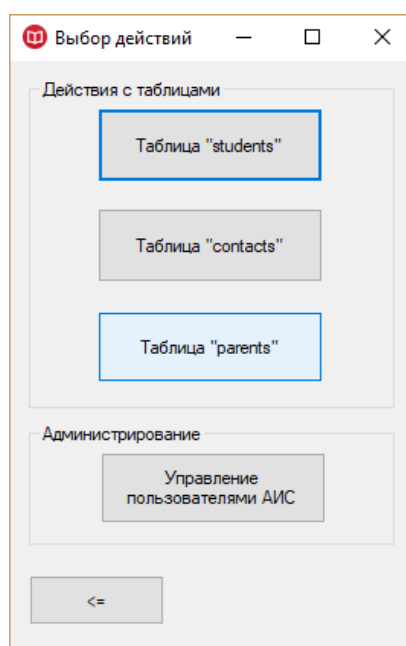


Рисунок 3.8 – Форма администрирования АИС

При выборе действий с таблицей пользователь переходит на форму с редактированием выбранной таблицы. В качестве примера рассмотрим таблицу students (рисунок 3.9). При успешном удалении пользователя из базы данных выводится соответствующее сообщение, а также удаляются связанные записи в таблицах contacts и parents (рисунок 3.10). При некорректном вводе или обновлении данных в таблицу аналогично выводится сообщение (рисунок 3.11). Фильтрация ввода осуществляется во всех полях таблицы.

ID	name	score	priority	form_study	major	original	form_pay	date
1	Петров Симон Г...	4,5	да	очная	ПКС	оригинал	бюджет	09.12.2017
2	Соколова Диан...	5	нет	очная	ИБАС	копия	бюджет	09.12.2017
3	Русина Фаина ...	4,8	нет	очная	ИБАС	оригинал	бюджет	09.12.2017
4	Быкова Галина ...	4,1	да	очная	ПКС	оригинал	бюджет	09.12.2017
5	Остимчук Кири...	3,6	да	очная	ПКС	оригинал	договор	09.12.2017
6	Лебедев Архип ...	4,6	да	очная	ПКС	оригинал	бюджет	09.12.2017
8	Егоров Харлам...	5	нет	заочная	ПКС	копия	бюджет	09.12.2017
9	Егоров Фёдор ...	3,6	да	очная	ИБАС	оригинал	договор	09.12.2017
10	Герасимова Ад...	4,2	да	очная	ПКС	копия	договор	09.12.2017
11	Иванова Людм...	3,65	да	очная	ПКС	оригинал	договор	09.12.2017
12	Алексеев Влад...	4,1	да	очная	ИБАС	оригинал	бюджет	09.12.2017

Действия с базой данных

Удаление из БД Добавление в БД Обновление БД

Рисунок 3.9 – Форма редактирования с таблицей students

ID	name	score	priority	form_study	major	original	form_pay	date
1	Петров Симон Г...	4,5	да	очная	ПКС	оригинал	бюджет	09.12.2017
2	Соколова Диан...	5	нет	очная	ИБАС	копия	бюджет	09.12.2017
3	Русина Фаина ...	4,8	нет	очная	ИБАС	оригинал	бюджет	09.12.2017
4	Быкова Галина ...	4,1	да	очная	ПКС	оригинал	бюджет	09.12.2017
5	Остимчук Кири...	3,6	да	очная	ПКС	оригинал	договор	09.12.2017
6	Лебедев Архип ...	4,6	да	очная	ПКС	оригинал	бюджет	09.12.2017
9	Егоров Фёдор ...	3,6	да	очная	ИБАС	оригинал	договор	09.12.2017
10	Герасимова Ад...	4,2	да	очная	ПКС	копия	договор	09.12.2017
11	Иванова Людм...	3,65	да	очная	ПКС	оригинал	договор	09.12.2017
12	Алексеев Влад...	4,1	да	очная	ИБАС	оригинал	бюджет	09.12.2017
13	Андропова Дар...	4,95	нет	очная	ИБАС	копия	бюджет	09.12.2017

Успешное удаление данных

OK

Действия с базой данных

Удаление из БД Добавление в БД Обновление БД

Рисунок 3.10 – Сообщение об успешном удалении данных из таблицы students

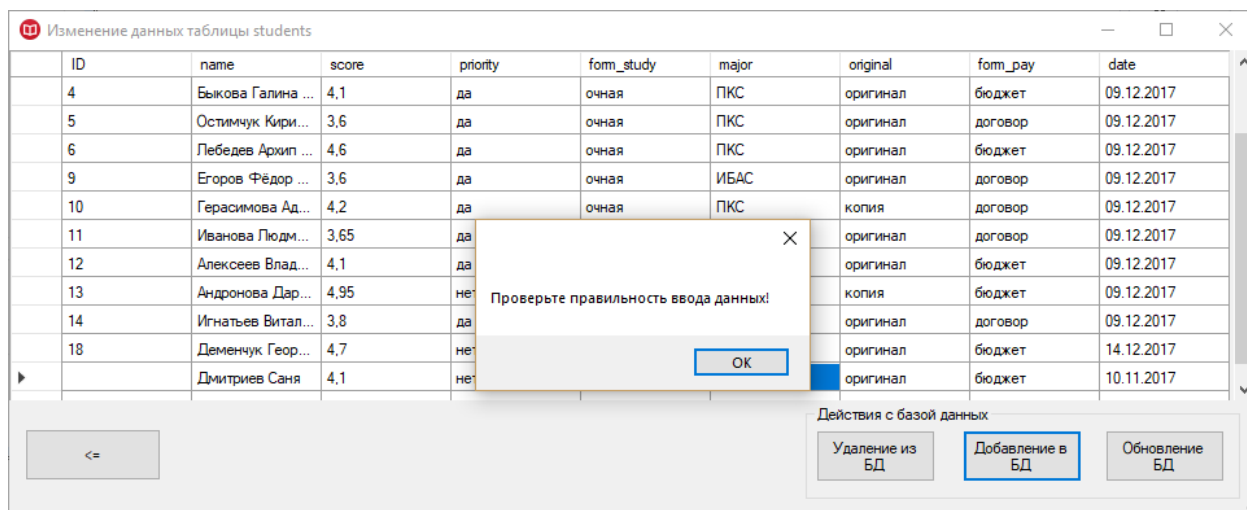


Рисунок 3.11 – Сообщение о некорректном вводе данных

При выборе управления пользователями АИС в меню действий, пользователь переходит на форму действий с пользователями АИС (рисунок 3.12). При переходе на форму добавления нового пользователя и при вводе пары логина и пароля, происходит проверка на дубликат логина/пароля (рисунок 3.13 – 3.14).

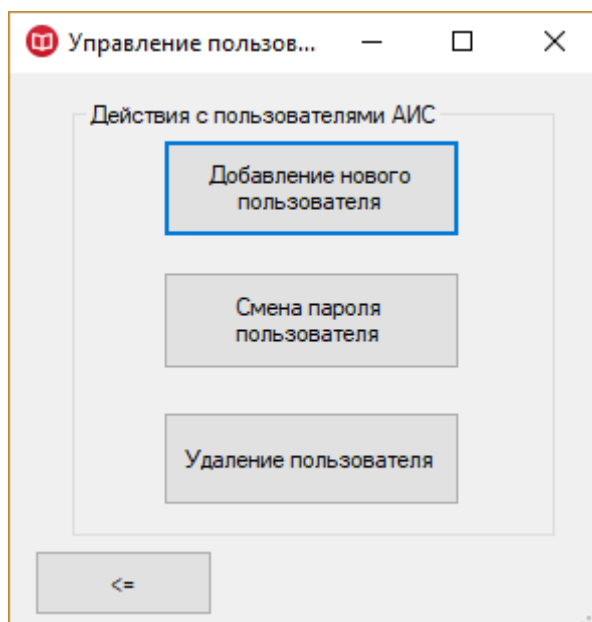


Рисунок 3.12 – Форма действий с пользователями АИС

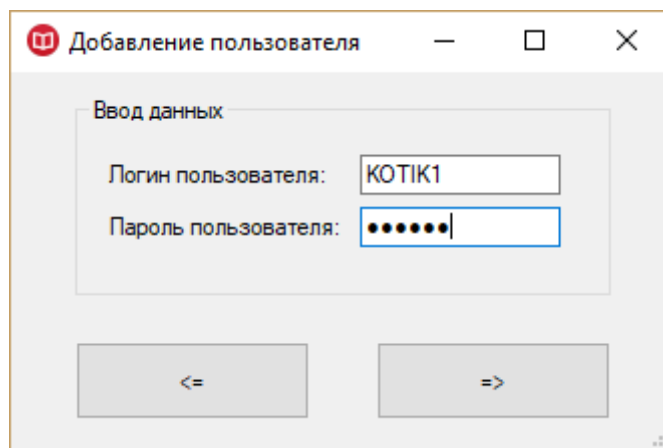


Рисунок 3.13 – Форма добавления нового пользователя АИС

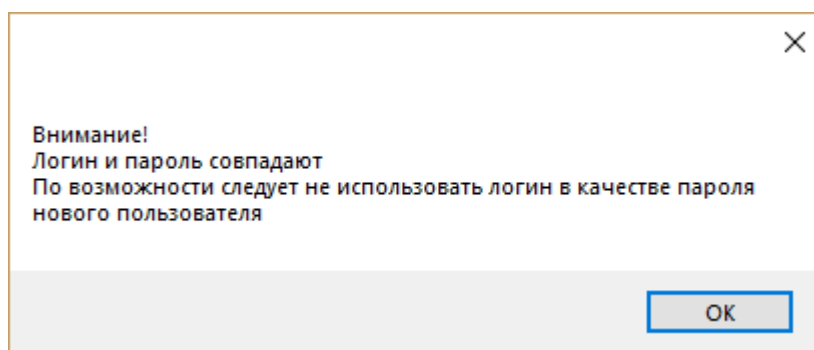


Рисунок 3.14 – Предупреждение о совпадении логина/пароля

При переходе на форму изменения пароля пользователя АИС (рисунок 3.15) и при вводе пары логин/старый пароль, а также нового пароля также происходит ряд проверок: проверка на существование пользователя (рисунок 3.16), проверка на верную пару логина/пароля (рисунок 3.17), при успешном выполнении задачи выводится соответствующее сообщение (рисунок 3.18).

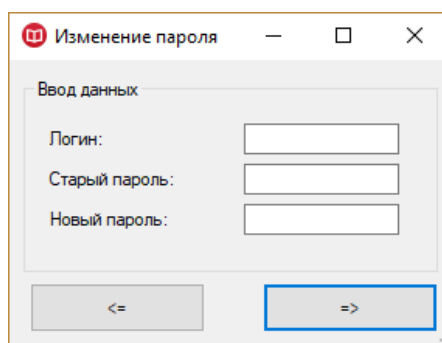


Рисунок 3.15 – Форма изменения пароля АИС

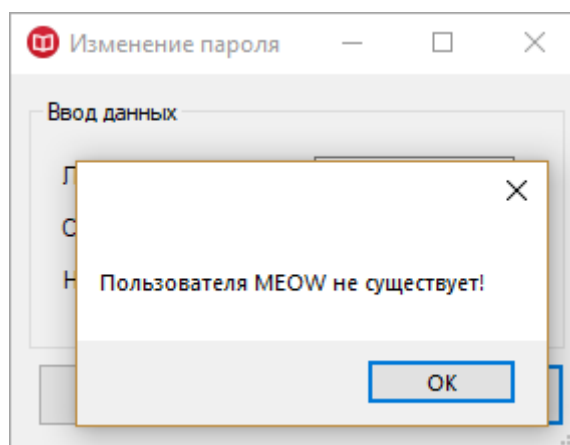


Рисунок 3.16 – Проверка на существование пользователя

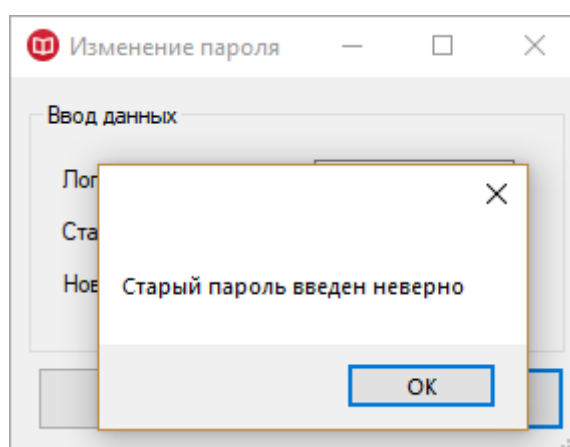


Рисунок 3.17 – Проверка на верную пару логин/старый пароль

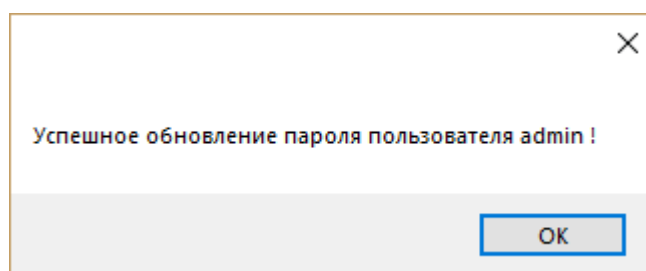


Рисунок 3.18 – Сообщение об успешном обновлении пароля пользователя АИС

При переходе на форму удаления пользователя АИС (рисунок 3.19) требуется ввод логина пользователя, пароль пользователя и мастер-пароль администратора, на них же и осуществляется проверка ввода. (рисунки 3.20 – 3.23).

Удаление пользователя

Ввод данных

Логин пользователя:

Пароль пользователя:

Мастер-пароль:

<= =>

Рисунок 3.19 – Форма удаления пользователя АИС

Введенного пользователя нет в системе!

ОК

Рисунок 3.20 – Сообщение о неверном логине пользователя АИС

Неверный пароль пользователя АИС!

ОК

Рисунок 3.21 – Сообщение о неверном пароле пользователя АИС

Мастер-пароль введен неверно!

ОК

Рисунок 3.22 – Сообщение о неверном мастер-пароле администратора АИС

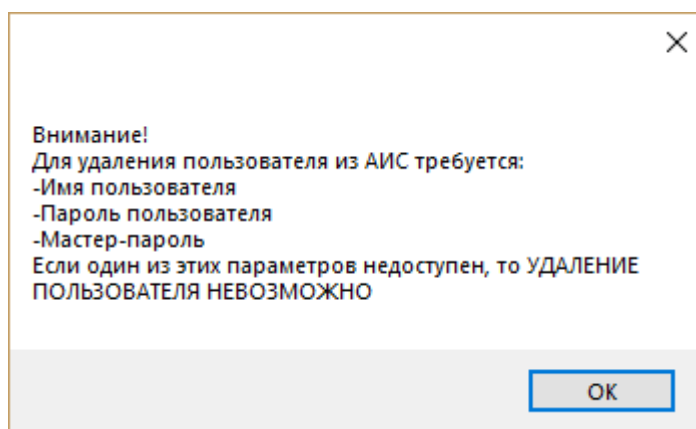


Рисунок 3.23 – Предупреждение администратора о запрашиваемых полях при возникновении каких-либо ошибок

3.1.4 Входные и выходные данные

Входные и выходные данные представлены в приложении А

3.1.5 Сообщения

В режиме администратора существует лишь один тип сообщений – информационное сообщение (рисунок 3.24).

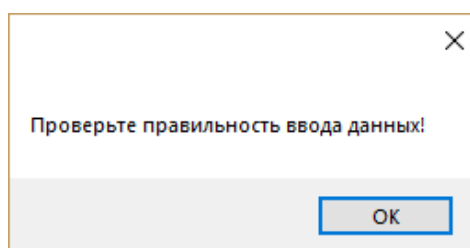


Рисунок 3.24 – Сообщение о некорректном вводе данных

3.2 Руководство пользователя

3.2.1 Назначение программы

Программа предназначена для сотрудников приёмной комиссии колледжа для упрощения и оптимизации приема абитуриентов.

3.2.2 Условия выполнения программы

Минимальные требования к электронной вычислительной машине:

- Процессор: Intel Celeron или аналогичный AMD;
- Частота: 1500 МГц;
- Оперативная память: 512 Мб;
- Клавиатура и мышь.

Рекомендуемые требования к Электронной вычислительной машине:

- Процессор: Intel Pentium;
- Частота: 2000 MHz;
- Оперативная память: 1 Гб;
- Клавиатура и мышь.

3.2.3 Выполнение программы

При первом запуске программы Вам требуется войти с парой логина/пароля, выданным администратором программы (рисунок 3.25). Все данные конфиденциальны и зашифрованы с помощью хеша MD5.

При правильной паре логина и пароля выводится сообщение об успешной авторизации (рисунок 3.26), в противном случае выводится сообщение о неверном пароле (рисунок 3.27).

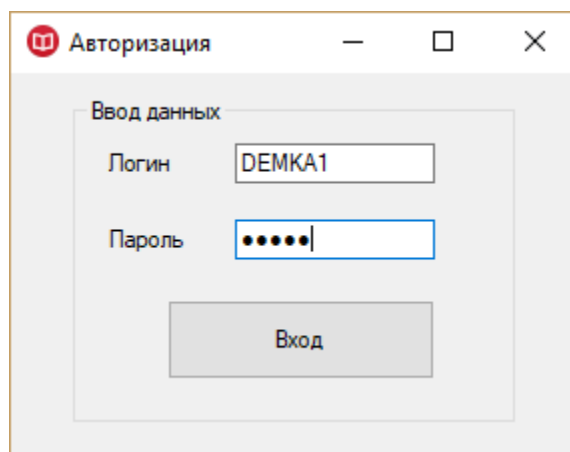


Рисунок 3.25 – Форма авторизации пользователя

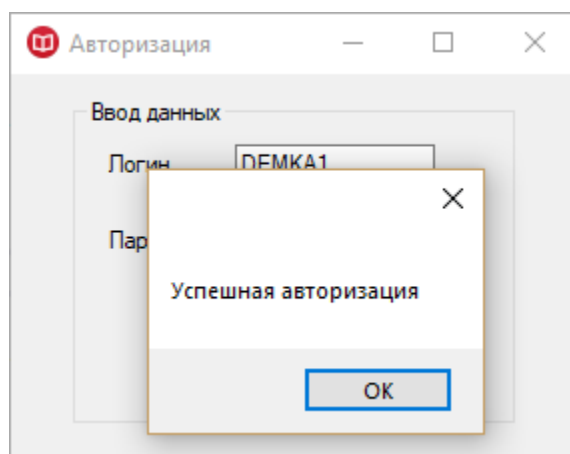


Рисунок 3.26 – Сообщение об успешной авторизации пользователя

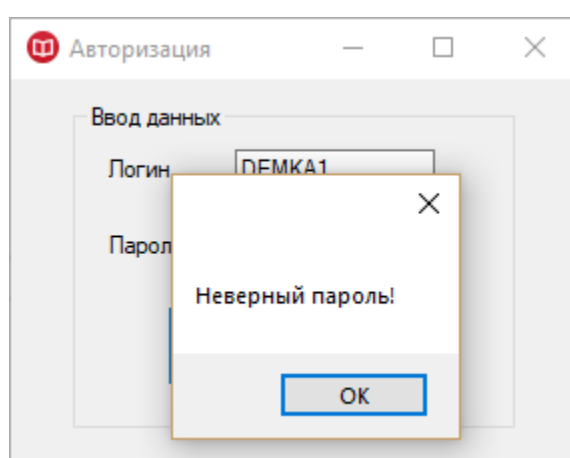


Рисунок 3.27 – Сообщение о неправильной паре логина/пароля

В случае успешной авторизации вам будет доступна главная форма действий с АИС (рисунок 3.28).

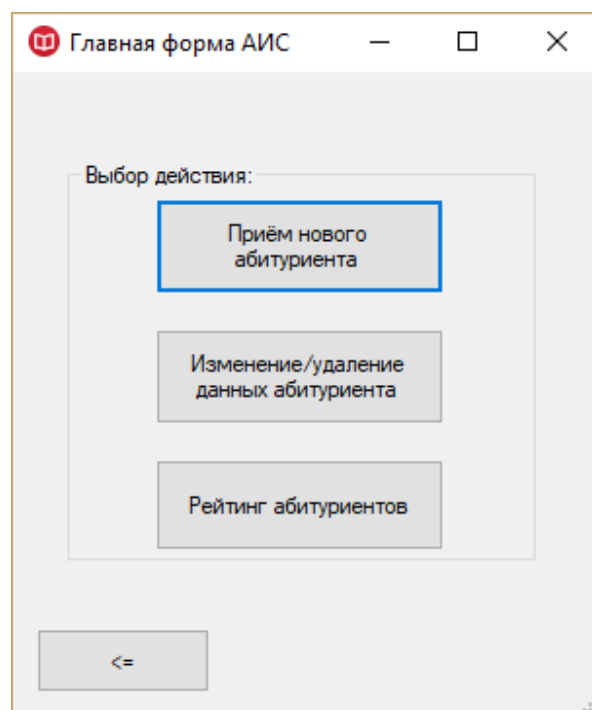


Рисунок 3.28 – Главная форма АИС

При нажатии на кнопку приема нового абитуриента выполняется переход на форму ввода основной информации об абитуриенте колледжа (рисунок 3.29). При некорректном вводе выводится соответствующее сообщение (рисунок 3.30). Также происходит фильтрация ФИО, среднего балла и направления. Существует проверка на отмеченные RadioButton (тип аттестата, форма обучения, приоритет, форма оплаты) (рисунок 3.31).

Рисунок 3.29 – Форма ввода основной информации об абитуриенте

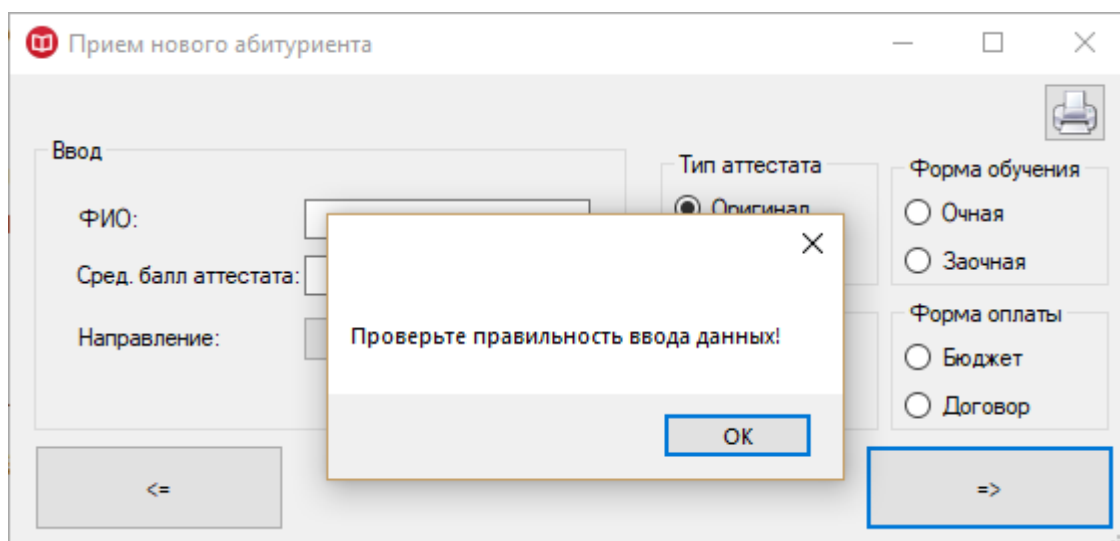


Рисунок 3.30 – Сообщение о некорректном вводе

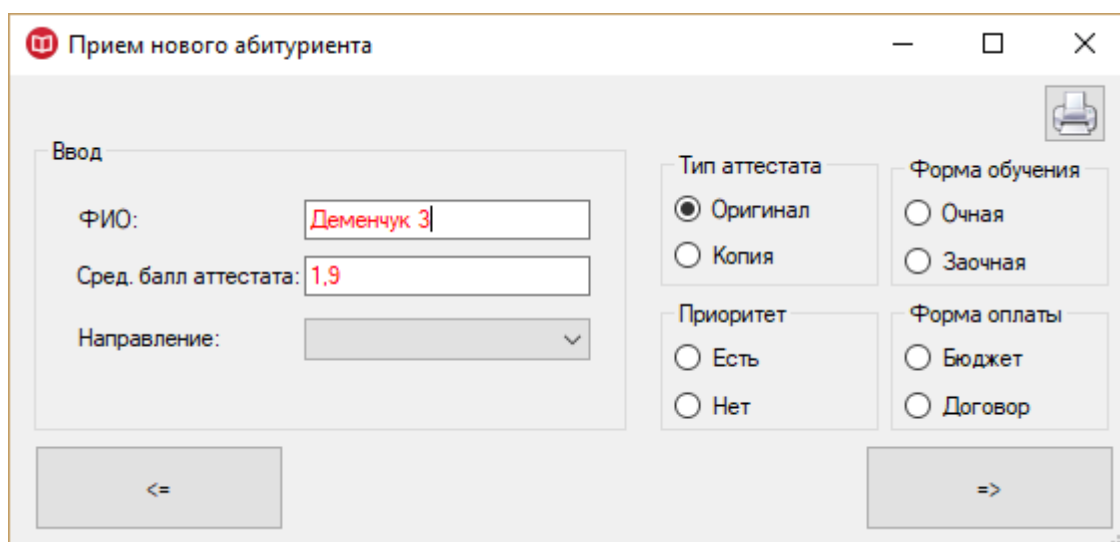


Рисунок 3.31 – Пример фильтрации на форме

При корректном вводе данных программа предлагает пользователю ввести дополнительную информацию о контактных данных абитуриента (рисунок 3.32). При условии, что пользователь соглашается с вводом дополнительной информации об абитуриенте, открывается форма ввода контактной информации абитуриента (рисунок 3.33). В данной форме происходит фильтрация e-mail и телефона (рисунки 3.34 – 3.35). При корректном вводе данных программа предлагает пользователю ввести контактные данные родителей (рисунок 3.36)

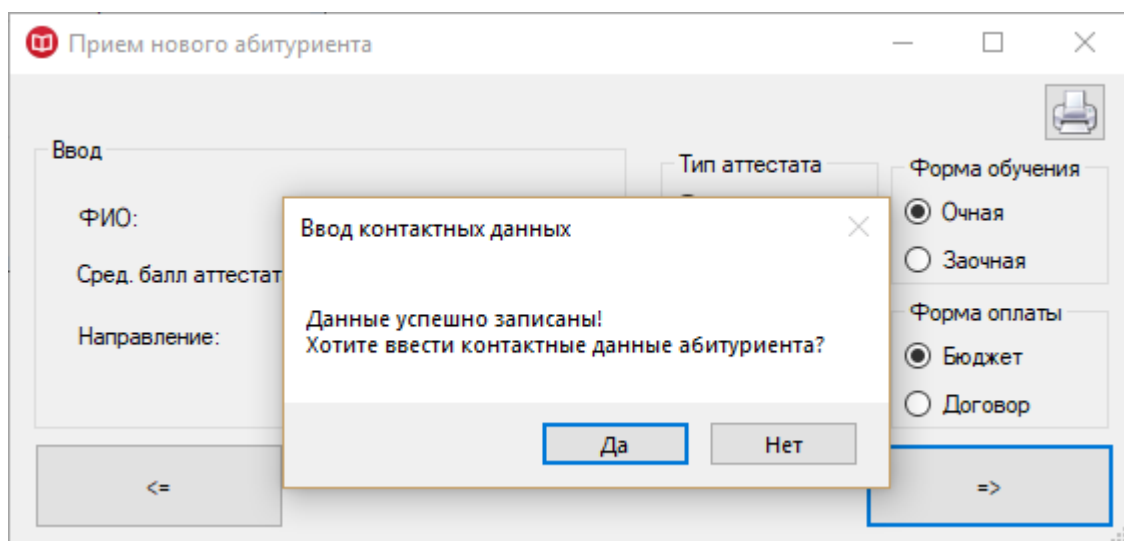


Рисунок 3.32 – Пример фильтрации на форме

Рисунок 3.33 – Форма ввода дополнительной информации об абитуриенте

Рисунок 3.34 – Пример фильтрации E-mail

Деменчук Георгий Макс...

Контактная информация абитуриента

Адрес: ул Ленина, д 4 корп. 2, кв 5

E-mail: demka1@mail.ru

Телефон: 794596454

<=

=>

Рисунок 3.35 – Пример фильтрации телефона

Деменчук Георгий Макс...

Контактная информация абитуриента

Контактные данные родителей

Контактные данные успешно записаны!
Хотите ввести контактные данные родителей?

Да Нет

Рисунок 3.36 – Предложение о заполнении контактных данных родителей.

При условии, что пользователь соглашается с вводом контактной информации о родителях, открывается форма ввода контактной информации абитуриента (рисунок 3.37). В данной форме происходит фильтрация ФИО, e-mail и телефона (рисунок 3.38).

Деменчук Георгий М...

Контактная информация родителей:

ФИО Деменчук Максим

Адрес: ул Ленина д 20 к 1 кв 281

E-mail: m@mail.com

Телефон: 79054054074

<= =>

Рисунок 3.37 – Форма контактных данных родителей

Деменчук Георгий М...

Контактная информация родителей:

ФИО Деменчук Максим 2

Адрес: ул Ленина д 20 к 1 кв 281

E-mail: max

Телефон: 7953

<= =>

Рисунок 3.38 – Форма контактных данных родителей

Если данные верны, то пользователю поступает предложение о формировании общего отчета об абитуриенте (рисунок 3.39). Если пользователь соглашается с формированием отчета, открывается форма конфигурации отчета об абитуриенте (рисунок 3.40). Пример выходного документа представлен на рисунке 3.41.

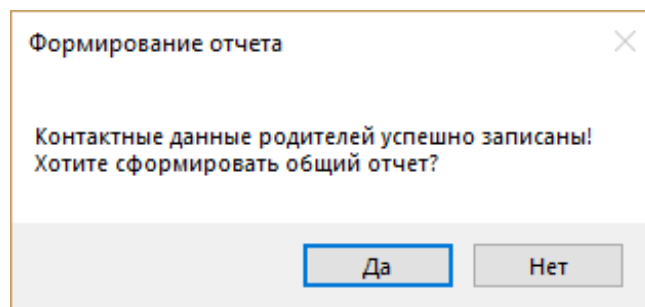


Рисунок 3.39 – Предложение о формировании общего отчета об абитуриенте

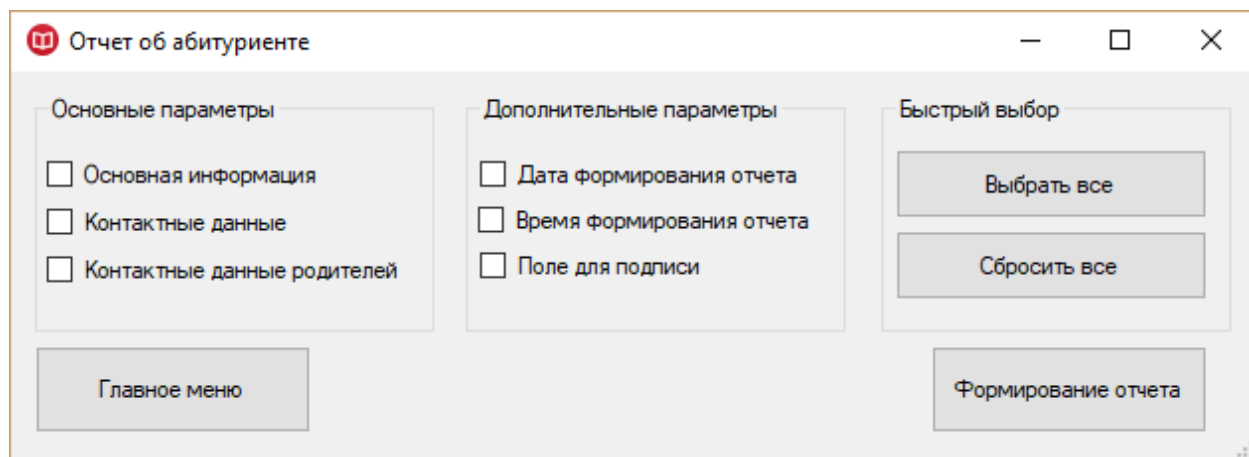


Рисунок 3.40 – Форма конфигурации отчета об абитуриенте

Справка о приеме документов в КИП при ФУ РФ

Основная информация:

ФИО: Деменчук Георгий Максимович

Средний балл аттестата: 4,7

Направление: ПКС

Форма оплаты: бюджет

Тип аттестата: оригинал

Форма обучения: очная

Приоритет: нет

Контактные данные абитуриента:

Адрес: ул Ленина, д 4 корп. 2, кв 58

E-mail: demka1@mail.ru

Телефон: 79459896454

Контактные данные родителей абитуриента:

ФИО: Деменчук Максим

Адрес: ул Ленина д 20 к 1 кв 281

E-mail: m@mail.com

Телефон: 79054054074

Дата: 14.12.2017

Время: 18:09

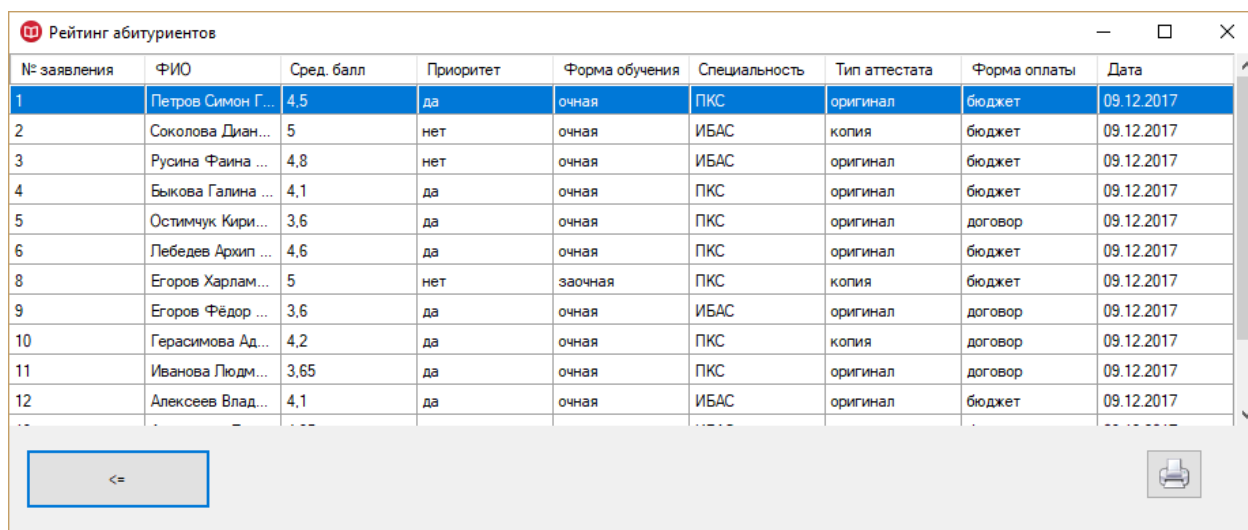
Подпись: _____

Расшифровка: _____

Рисунок 3.41 – Пример сформированного отчета

После формирования отчета программа автоматически переходит обратно на форму главного меню АИС (рисунок 3.4). При нажатии нижней кнопки формы выполняется открытие формы рейтинга абитуриентов (рисунок 3.42), возможна сортировка по любому столбцу. При желании пользователя возможен переход к мастеру формирования отчетов (рисунок 3.43), где идет вызов формы печати общего рейтинга (рисунки 3.44 – 3.45) абитуриента или отчета о

конкретном абитуриенте (рисунок 3.46) (для этого предварительно необходимо выбрать абитуриента на элементе GridTable на форме рейтинга абитуриентов).



№ заявления	ФИО	Сред. балл	Приоритет	Форма обучения	Специальность	Тип аттестата	Форма оплаты	Дата
1	Петров Симон Г...	4,5	да	очная	ПКС	оригинал	бюджет	09.12.2017
2	Соколова Диан...	5	нет	очная	ИБАС	копия	бюджет	09.12.2017
3	Русина Фаина ...	4,8	нет	очная	ИБАС	оригинал	бюджет	09.12.2017
4	Быкова Галина ...	4,1	да	очная	ПКС	оригинал	бюджет	09.12.2017
5	Остимчук Кири...	3,6	да	очная	ПКС	оригинал	договор	09.12.2017
6	Лебедев Архип ...	4,6	да	очная	ПКС	оригинал	бюджет	09.12.2017
8	Егоров Харлам...	5	нет	заочная	ПКС	копия	бюджет	09.12.2017
9	Егоров Фёдор ...	3,6	да	очная	ИБАС	оригинал	договор	09.12.2017
10	Герасимова Ад...	4,2	да	очная	ПКС	копия	договор	09.12.2017
11	Иванова Людм...	3,65	да	очная	ПКС	оригинал	договор	09.12.2017
12	Алексеев Влад...	4,1	да	очная	ИБАС	оригинал	бюджет	09.12.2017

Рисунок 3.42 – Форма рейтинга абитуриентов

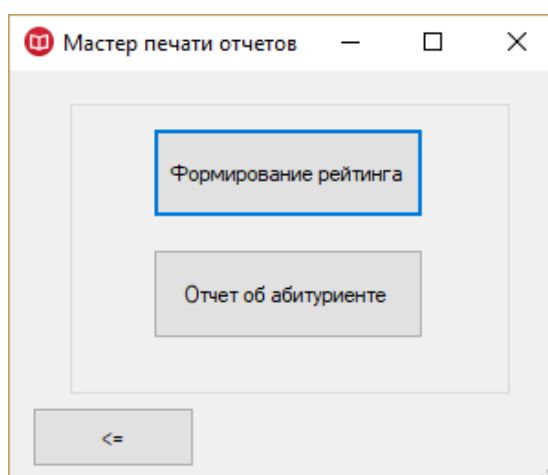


Рисунок 3.43 – Мастер печати отчетов

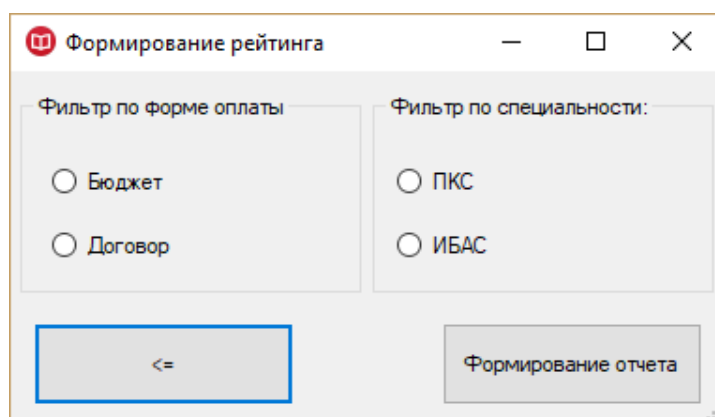



Рисунок 3.44 – Формирование общего рейтинга абитуриентов

Рейтинг абитуриентов

№	ФИО	Средний балл	Приоритет	Форма обучения	Специальность	Тип аттестата	Форма оплаты	Дата
1	Егоров Харлампий Макарович	5	нет	заочная	ПКС	копия	бюджет	09.12.2017
2	Лебедев Архип Денисович	4,6	да	очная	ПКС	оригинал	бюджет	09.12.2017
3	Петров Симон Геннадиевич	4,5	да	очная	ПКС	оригинал	бюджет	09.12.2017
4	Герасимова Ада Ефимовна	4,2	да	очная	ПКС	копия	договор	09.12.2017
5	Быкова Галина Леонидовна	4,1	да	очная	ПКС	оригинал	бюджет	09.12.2017
6	Иванова Людмила Павловна	3,65	да	очная	ПКС	оригинал	договор	09.12.2017
7	Остимчук Кирилл Кириллович	3,6	да	очная	ПКС	оригинал	договор	09.12.2017
8	Леонова Алевтина Николаевна	3,15	нет	очная	ПКС	копия	договор	09.12.2017

Рисунок 3.45 – Пример сформированного отчета

 Отчет об абитуриенте (Петров Симон Геннадиевич)

Основные параметры

- ☒ Основная информация
- ☒ Контактные данные
- ☒ Контактные данные родителей

Дополнительные параметры

- ☒ Дата формирования отчета
- ☒ Время формирования отчета
- ☒ Поле для подписи

Быстрый выбор

Выбрать все

Сбросить все

<=

Формирование отчета

Рисунок 3.46 – Формирование отчета об абитуриенте

3.2.4 Сообщение пользователю

В режиме пользователя в программе существует два типа сообщений:

1. Информационное сообщение;
2. Сообщение с выбором.

Например, при успешной авторизации или при некорректном заполнении основных данных абитуриента выводится соответствующее сообщение первого типа (рисунки 3.47 – 3.48). При успешной записи основной информации об абитуриенте пользователю поступает сообщение второго типа о заполнении контактной информации абитуриента (рисунок 3.49), также при выходе из пользователя из АИС, система уточняет действительно ли он хочет выйти из системы (рисунок 3.50).

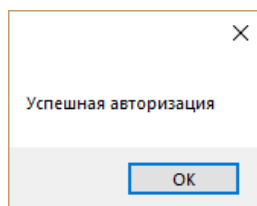


Рисунок 3.47 – Сообщение об успешной авторизации пользователя

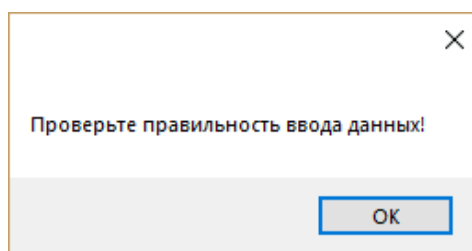


Рисунок 3.48 – Сообщение о некорректном вводе данных

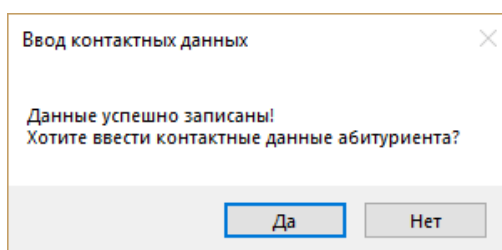


Рисунок 3.49 – Предложение о заполнении контактных данных абитуриента

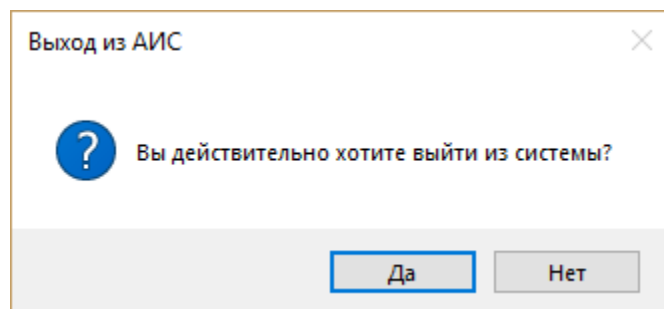


Рисунок 3.50 – Уточнение о выходе пользователя из АИС

ЗАКЛЮЧЕНИЕ

В данной курсовой представлена автоматизированная информационная система для средних профессиональных образовательных учреждений.

При разработке проекта большое внимание было уделено безопасности и предотвращению несанкционированных действий со стороны пользователей: разделение прав пользователей и администраторов АИС при помощи мастер-пароля, а также шифрование пар логинов/паролей посредством MD5 хеш-сумм строк, конвертированных в формат стандарта Base64.

Также за счет C++/CLI с Windows Forms стало возможным реализовать интуитивно понятный интерфейс для взаимодействия с пользователем, следовательно, для использования программы пользователю достаточно знать лишь основы работы с компьютером.

Отдельно стоит отметить динамическое формирование отчетов в Microsoft Office Word: пользователю предоставляется возможность выбора пунктов для вывода.

И еще одно достоинство – использование компактной встраиваемой реляционной база данных SQLite3, которая идеально подходит для портируемых и не предназначенных для масштабирования приложений.

Цели и задачи, поставленные при выполнении курсового проекта, выполнены с соблюдением всех предъявленных требований.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

Стандарты

1. ГОСТ 7.1. – 2003. Библиографическая запись. Библиографическое описание. Общие требования и правила составления. – М.: ИПК Издательство стандартов, 2004. – 169 с.
2. ГОСТ 7.32 – 2001. Система стандартов по информации, библиотечному и издательскому делу. Отчет о научно-исследовательской работе. Структура и правила оформления. – М.: ИПК Издательство стандартов, 2001. – 21 с.
3. ГОСТ 7.82 – 2001. Библиографическая запись. Библиографическое описание электронных ресурсов. Общие требования и правила составления. – М.: ИПК Издательство стандартов, 2001. – 21 с.
4. Единая система программной документации. – М.: Стандартинформ, 2005. – 128 с.

Интернет-ресурсы

5. Язык программирования C#: учебный курс А. Фридмана. – [Электронный ресурс]. – Режим доступа: <http://www.intuit.ru/studies/courses/17/17/info>
6. Миронова А.И. К вопросу об эффективности применения баз данных в туризме // Современные научные исследования и инновации. 2015. [Электронный ресурс]. <http://web.snauka.ru/issues/2015/03/50794>
7. Сфера применения БД. Студопедия. [Электронный ресурс]. <http://studopedia.org/8-52848.html>
8. Автоматизированная информационная система. Студопедия. [Электронный ресурс]. http://studopedia.ru/15_26125_avtomatizirovannaya-informatsionnaya-sistema.html

9. Microsoft Visual Studio. TADVISER. [Электронный ресурс].
<http://tadviser.ru/a/54258>
10. C++. Progopedia. [Электронный ресурс]. <http://progopedia.ru/language/c-plus-plus/>

ПРИЛОЖЕНИЕ А

ВХОДНАЯ/ВЫХОДНАЯ ИНФОРМАЦИЯ

Входная и выходная информация представлена в таблице 1.

Таблица А1 – Входные/выходные данные

Тип	Форма	Идентификатор	Наименование	Тип данных
Входные	StafLoginForm.h	LoginBox	Логин пользователя АИС	Строковый
Входные	StafLoginForm.h	PasswordBox	Пароль пользователя АИС	Строковый
Входные	InputForm.h	FIOBox	ФИО абитуриента	Строковый
Входные	InputForm.h	ScoreBox	Средний балл абитуриента	Вещественны й
Входные	InputForm.h	MajorComboBox	Направление абитуриента	Строковый
Входные	InputForm.h	OriginalBox	Выбор типа аттестата	Строковый
Входные	InputForm.h	PriorityBox	Выбор приоритета	Строковый
Входные	InputForm.h	StudyformBox	Выбор формы обучения	Строковый
Входные	InputForm.h	FormPayGroupBox	Выбор формы оплаты	Строковый
Входные	MoreStudentInfoForm.h	AdressBox	Адрес абитуриента	Строковый
Входные	MoreStudentInfoForm.h	EmailBox	Электронная почта абитуриента	Строковый
Входные	MoreStudentInfoForm.h	PhoneBox	Телефон абитуриента	Строковый
Входные	ParentsContactForm.h	AdressBox	Checkbox для дублирования адреса	Логический

Входные	ParentsContactForm.h	ParentFIOBox	ФИО родителя абитуриента	Строковый
Входные	ParentsContactForm.h	ParentAdressBox	Адрес родителя абитуриента	Строковый
Входные	ParentsContactForm.h	ParentEmailBox	Электронная почта родителя абитуриента	Строковый
Входные	ParentsContactForm.h	ParentPhoneBox	Телефон родителя абитуриента	Строковый
Входные	FinalPrintForm.h	MainInfoCheckBox	Вывод на печать основной информации	Логический
Входные	FinalPrintForm.h	StudentContactsCheckBox	Вывод на печать контактов абитуриента	Логический
Входные	FinalPrintForm.h	ParentsContactsCheckBox	Вывод на печать контактов родителей абитуриента	Логический
Входные	FinalPrintForm.h	DateCheckBox	Вывод на печать даты формирования отчета	Логический
Входные	FinalPrintForm.h	TimeCheckBox	Вывод на печать времени формирования отчета	Логический
Входные	FinalPrintForm.h	WriterCheckBox	Вывод на печать поля для подписи	Логический
Входные	PasswordChecker.h	PasswordCheckerBox	Ввод мастер-пароля	Строковый
Входные	AddNewStaffForm.h	NewStaffLoginBox	Логин нового пользователя АИС	Строковый

Входные	AddNewStaffForm.h	NewStaffPasswordBox	Пароль нового пользователя АИС	Строковый
Входные	PasswordStaffForm.h	StaffLoginBox	Логин для изменения пароля АИС	Строковый
Входные	PasswordStaffForm.h	StaffOldPasswordBox	Старый пароль для изменения пароля АИС	Строковый
Входные	PasswordStaffForm.h	StaffNewPasswordBox	Новый пароль для изменения пароля АИС	Строковый
Входные	RemoveStaffForm.h	StaffLoginBox	Логин для удаления пользователя АИС	Строковый
Входные	RemoveStaffForm.h	StaffPasswordBox	Пароль для удаления пользователя АИС	Строковый
Входные	RemoveStaffForm.h	MasterPasswordBox	Мастер-пароль для удаления пользователя АИС	Строковый
Выходные	DBChangerForm.h	dataGridView1	Вывод таблицы students на экран	—
Выходные	ContactsDBChangerForm.h	dataGridView1	Вывод таблицы contacts на экран	—
Выходные	ParentsDBChangerForm.h	dataGridView1	Вывод таблицы parents на экран	—
Выходные	RatingForm.h	dataGridView1	Вывод форматированной таблицы students на экран	—
Входные	PrintForm.h	PayRadioButton1	Выборка по форме оплаты «бюджет»	Логический

Входные	PrintForm.h	PayRadioButton2	Выборка по форме оплаты «договор»	Логический
Входные	PrintForm.h	MajorRadioButto n1	Выборка по специальности «ПКС»	Логический
Входные	PrintForm.h	MajorRadioButto n2	Выборка по специальности «ПКС»	Логический

ПРИЛОЖЕНИЕ Б

ЛИСТИНГ ПРОГРАММЫ

Модуль «GlobalClass.h»

```
#include <regex>
```

```
#pragma once
```

```
namespace Globals
```

```
{
```

```
    using namespace System;
```

```
    using namespace System::Text;
```

```
    using namespace System::Security::Cryptography;
```

```
    public ref class GlobalClass
```

```
    {
```

```
    public:
```

```
        static String^ SQLGlobalPatch;
```

```
        static String^ MasterGlobalPassword;
```

```
        static String^ getMD5String(String^ inputString)
```

```
        {
```

```
            array<Byte>^ byteArray = Encoding::ASCII->GetBytes(inputString);
```

```
            MD5CryptoServiceProvider^ md5provider = gcnew MD5CryptoServiceProvider();
```

```
            array<Byte>^ byteArrayHash = md5provider->ComputeHash(byteArray);
```

```
            return BitConverter::ToString(byteArrayHash);
```

```
        }
```

```
        static String^ fromBase64(String^ from)
```

```
        {
```

```
            System::Text::UTF8Encoding^ encoding = gcnew System::Text::UTF8Encoding();
```

```
            array<unsigned char>^ base64 = System::Convert::FromBase64String(from);
```

```
            return encoding->GetString(base64);
```

```
        }
```

```
        static String^ toBase64(String^ from)
```



```

        {
            System::Text::UTF8Encoding^ encoding = gcnew System::Text::UTF8Encoding();
            System::String^ base64 = System::Convert::ToBase64String(encoding-
>GetBytes(from));
            return base64->Substring(0, base64->Length - 0);
        }
        static bool is_valid_number(const std::string& number)
        {
            static const std::string AllowedChars = "0123456789";
            for (auto numberChar = number.begin();
                numberChar != number.end(); ++numberChar)
                if (AllowedChars.end() == std::find(
                    AllowedChars.begin(), AllowedChars.end(), *numberChar)
                )
                {
                    return false;
                }
            return true;
        }
        static bool is_valid_email(const std::string& email)
        {
            const std::regex pattern
            ("(\\w+)(\\.|_|_)?(\\w*)@(\\w+)(\\.\\w+)+");
            return std::regex_match(email, pattern);
        }
    };
}

```

Модуль «AddNewStaffForm.h»

```
#pragma endregion
```

```

private: void EmptyAllTextBox() {
    NewStaffLoginBox->Text = "";
    NewStaffPasswordBox->Text = "";
}

```

```

private: void NewStaffAddSQL(String^ login_str, String^ password_str) {

    SQLiteConnection ^db = gcnew SQLiteConnection();

    String^ MD5login_str = GlobalClass::getMD5String(GlobalClass::toBase64(login_str));

    String^ MD5password_str = GlobalClass::getMD5String(GlobalClass::toBase64(password_str));
    String^ SQL_STRING = "INSERT INTO staff VALUES(NULL,'" + MD5login_str + "','" +
MD5password_str + "')";

    db->ConnectionString = GlobalClass::SQLGlobalPatch;

    db->Open();

    SQLiteCommand ^cmdInsertValue = db->CreateCommand();
    cmdInsertValue->CommandText = SQL_STRING;

    cmdInsertValue->ExecuteNonQuery();

    db->Close();

    MessageBox::Show("Успешное добавление нового пользователя АИС");

    this->Hide();

}

private: bool StaffPasswordSQLChecker(String^ password_str) {

    SQLiteConnection ^db = gcnew SQLiteConnection();

    String^ MD5password_str = GlobalClass::getMD5String(GlobalClass::toBase64(password_str));

    String^ BufValidationStr = "DEMKA";
    String^ SQL_STRING = "SELECT * FROM staff WHERE password ='" + MD5password_str +
    "','";

    db->ConnectionString = GlobalClass::SQLGlobalPatch;

    db->Open();

    SQLiteCommand ^cmdSelect = db->CreateCommand();

    cmdSelect->CommandText = SQL_STRING;

    SQLiteDataReader ^data = cmdSelect->ExecuteReader();

    while (data->Read())

        BufValidationStr = data->GetValue(0)->ToString();

    db->Close();

    if (BufValidationStr == "DEMKA")

        return true;

    return false;

}

private: bool StaffLoginSQLChecker(String^ login_str) {

```

```

SQLiteConnection ^db = gcnew SQLiteConnection();

String^ MD5login_str = GlobalClass::getMD5String(GlobalClass::toBase64(login_str));

String^ BufValidationStr = "DEMKA";
String^ SQL_STRING = "SELECT * FROM staff WHERE login =" + MD5login_str + """;

db->ConnectionString = GlobalClass::SQLGlobalPatch;

db->Open();

SQLiteCommand ^cmdSelect = db->CreateCommand();

cmdSelect->CommandText = SQL_STRING;

SQLiteDataReader ^data = cmdSelect->ExecuteReader();

while (data->Read())

    BufValidationStr = data->GetValue(0)->ToString();

db->Close();

if (BufValidationStr == "DEMKA")

    return true;

return false;

}

private: System::Void AddStaffButton_Click(System::Object^ sender, System::EventArgs^ e)

{

    if ((NewStaffLoginBox->Text) == (NewStaffPasswordBox->Text))

        MessageBox::Show("Внимание!\nЛогин и пароль совпадают\nПо возможности  
следует не использовать логин в качестве пароля нового пользователя");

        if ((StaffLoginSQLChecker(NewStaffLoginBox->Text)) &&
            (StaffPasswordSQLChecker(NewStaffPasswordBox->Text)))

            NewStaffAddSQL(NewStaffLoginBox->Text, NewStaffPasswordBox->Text);

        else if (StaffLoginSQLChecker(NewStaffLoginBox->Text) == false)

            MessageBox::Show("Внимание!\nДанный логин уже используется в системе,  
добавление невозможно");

        else if (MessageBox::Show("Внимание!\nДанный пароль уже используется одним или  
несколькими пользователями АИС\nХотите продолжить?", "Дубликат пароля",
            MessageBoxButtons::YesNo, MessageBoxIcon::Question) == System::Windows::Forms::DialogResult::Yes)

            NewStaffAddSQL(NewStaffLoginBox->Text, NewStaffPasswordBox->Text);

```

```

        EmptyAllTextBox();
    }

private: System::Void ExitButton_Click(System::Object^ sender, System::EventArgs^ e) {
    this->Hide();
}

};
}

```

Модуль «ChoicePrintTypeForm.h»

#pragma endregion

```

public: String^ PublicStudentID;
public: String^ PublicStudentFIO;

private: System::Void MenuButton_Click(System::Object^ sender, System::EventArgs^ e) {
    this->Hide();
}
private: System::Void RatingButton_Click(System::Object^ sender, System::EventArgs^ e) {
    DEMKA::PrintForm^PrintForm_obj = gnew DEMKA::PrintForm();
    this->Hide();
    PrintForm_obj->ShowDialog();
}

private: System::Void ChoicePrintTypeForm_Load(System::Object^ sender, System::EventArgs^ e) {
}
private: System::Void ContactsTableButton_Click(System::Object^ sender, System::EventArgs^ e) {
    DEMKA::FinalPrintForm^FinalPrintForm_obj = gnew DEMKA::FinalPrintForm();

    FinalPrintForm_obj->PublicStudentID = PublicStudentID;
    this->Hide();
    FinalPrintForm_obj->ExitButton->Text = "<=";
    FinalPrintForm_obj->Text = "Отчет об абитуриенте (" + PublicStudentFIO + ")";
    FinalPrintForm_obj->ShowDialog();
}

};
}

```

Модуль «ContactsDBChangerForm.h»

```

String^ changer_fix;

array<String^>^ GridTableRow_array;

private: int valid_checker() {

    array<bool>^ BoolCheckArr;
    BoolCheckArr = gcnew array<bool>(5);
    for (int i = 0; i < BoolCheckArr->Length; i++) {
        BoolCheckArr[i] = false;
    }

    //Валидация e-mail
    if (GridTableRow_array[3] == "")
    {
        BoolCheckArr[3] = false;
    }
    else {
        BoolCheckArr[3] = true;

        msclr::interop::marshal_context oMarshalContext;

        const char* buf = oMarshalContext.marshal_as<const char*>(GridTableRow_array[3]);

        if (GlobalClass::is_valid_email(buf) == false)

            BoolCheckArr[3] = false;
    }

    //Валидация телефона
    if (GridTableRow_array[4] == "")
    {
        BoolCheckArr[4] = false;
    }
    else {
        BoolCheckArr[4] = true;

        msclr::interop::marshal_context oMarshalContext;

        const char* buf = oMarshalContext.marshal_as<const char*>(GridTableRow_array[4]);

        if (GlobalClass::is_valid_number(buf) == false)

            BoolCheckArr[4] = false;
    }

    //Другие валидации
    BoolCheckArr[2] = (GridTableRow_array[2] != "") ? true : false;
}

```

```

        for (int i = 2; i < BoolCheckArr->Length; i++)
            if (BoolCheckArr[i] == false)
                return 1;

        return 0;
    }

private: DataTable^ GetDataTable() {
    DataTable ^table;

    SQLiteConnection ^db = gcnew SQLiteConnection();
    db->ConnectionString = GlobalClass::SQLGlobalPatch;
    db->Open();
    SQLiteCommand ^cmdSelect = db->CreateCommand();
    cmdSelect->CommandText = "SELECT * FROM contacts;";
    SQLiteDataReader ^reader = cmdSelect->ExecuteReader();
    DataColumn ^column;
    DataRow ^row;
    table = gcnew DataTable();
    vector<String^>^ nameColumns = gcnew vector<String^>();
    for (int i = 0; i < (reader->FieldCount); i++) {
        nameColumns->push_back(reader->GetName(i));
        column = gcnew DataColumn(nameColumns->at(i), String::typeid);
        table->Columns->Add(column);
    }
    while (reader->Read()) {
        row = table->NewRow();
        for (int i = 0; i < (reader->FieldCount); i++) {
            row[nameColumns->at(i)] = reader->GetValue(i)->ToString();
            reader->GetValue(i)->ToString();
        }
        table->Rows->Add(row);
    }
    db->Close();
    return table;
}

```

```

private: System::Void ExitButton_Click(System::Object^ sender, System::EventArgs^ e) {
    this->Hide();
}

private: System::Void ContactsDBChangerForm_Load(System::Object^ sender, System::EventArgs^ e)
{
    dataGridView1->DataSource = GetDataTable();
}

private: System::Void RemoveBDBButton_Click(System::Object^ sender, System::EventArgs^ e) {
    int index = dataGridView1->CurrentCell->RowIndex;

    String^ ID = dataGridView1->Rows[index]->Cells["ID"]->Value->ToString();

    SQLiteConnection ^db = gcnew SQLiteConnection();
    db->ConnectionString = GlobalClass::SQLGlobalPatch;
    db->Open();

    SQLiteCommand ^cmdInsertValue = db->CreateCommand();
    cmdInsertValue->CommandText = "DELETE FROM contacts WHERE ID = " + ID + ";";
    cmdInsertValue->ExecuteNonQuery();
    db->Close();

    dataGridView1->DataSource = GetDataTable();

    MessageBox::Show("Успешное удаление данных");
}

private: System::Void AddDBButton_Click(System::Object^ sender, System::EventArgs^ e) {
    int index = dataGridView1->CurrentCell->RowIndex;

    GridTableRow_array = gcnew array<String^>(5);

    for (int i = 0; i < GridTableRow_array->Length; i++)
        GridTableRow_array[i] = dataGridView1->Rows[index]->Cells[i]->Value->ToString();

    if (valid_checker() == 0) {
        SQLiteConnection ^db = gcnew SQLiteConnection();

        db->ConnectionString = GlobalClass::SQLGlobalPatch;
        db->Open();
        SQLiteCommand ^cmdInsertValue = db->CreateCommand();
        cmdInsertValue->CommandText = "INSERT INTO contacts VALUES(NULL," +
GridTableRow_array[1] + "," + GridTableRow_array[2] +
        "," + GridTableRow_array[3] + "," + GridTableRow_array[4] + ")";

        cmdInsertValue->ExecuteNonQuery();
        db->Close();
    }
}

```

```

        MessageBox::Show("Успешная запись данных");
    }
    else
        MessageBox::Show("Проверьте правильность ввода данных!");
}

private: System::Void UpdateBDButton_Click(System::Object^ sender, System::EventArgs^ e) {
    int index = dataGridView1->CurrentCell->RowIndex;
    String^ ID = dataGridView1->Rows[index]->Cells["ID"]->Value->ToString();
    GridTableRow_array = gcnew array<String^>(5);
    for (int i = 0; i < GridTableRow_array->Length; i++)
        GridTableRow_array[i] = dataGridView1->Rows[index]->Cells[i]->Value->ToString();

    if (valid_checker() == 0) {
        SQLiteConnection ^db = gcnew SQLiteConnection();
        db->ConnectionString = GlobalClass::SQLGlobalPatch;
        db->Open();
        SQLiteCommand ^cmdInsertValue = db->CreateCommand();
        cmdInsertValue->CommandText = "UPDATE contacts SET student_id=" +
GridTableRow_array[1] + ",adress=" + GridTableRow_array[2] +
        "",e_mail=" + GridTableRow_array[3] + ",telephone=" + GridTableRow_array[4] + ""
WHERE ID =" + ID;
        cmdInsertValue->ExecuteNonQuery();
        db->Close();
        MessageBox::Show("Успешное обновление данных");
    }
    else
        MessageBox::Show("Проверьте правильность ввода данных!");
    dataGridView1->DataSource = GetDataTable();
}

private: System::Void backgroundWorker1_DoWork(System::Object^ sender,
System::ComponentModel::DoWorkEventArgs^ e) {
}
};
}

```

Модуль «DBChangerForm.h»


```
#pragma endregion
```

```
String^ changer_fix;
```

```
array<String^>^ GridTableRow_array;
```

```
private: int valid_checker() {
```

```
    array<bool>^ BoolCheckArr;
```

```
    BoolCheckArr = gcnew array<bool>(8);
```

```
    for (int i = 0; i < BoolCheckArr->Length; i++) {
```

```
        BoolCheckArr[i] = false;
```

```
    }
```

```
    //Валидация ФИО
```

```
    if (GridTableRow_array[1] == "")
```

```
    {
```

```
        BoolCheckArr[1] = false;
```

```
    }
```

```
    else {
```

```
        BoolCheckArr[1] = true;
```

```
        msclr::interop::marshal_context oMarshalContext;
```

```
        const char* buf = oMarshalContext.marshal_as<const char*>(GridTableRow_array[1]);
```

```
        for (int i = 0; i < System::Convert::ToInt32(strlen(buf)); i++) {
```

```
            if (iswdigit(buf[i])) {
```

```
                BoolCheckArr[1] = false;
```

```
            }
```

```
        }
```

```
    }
```

```
    //Валидация Сред. балла
```

```
    msclr::interop::marshal_context context;
```

```
    std::string buf_str = context.marshal_as<std::string>(GridTableRow_array[2]);
```

```
    double d;
```

```
    if (GridTableRow_array[2] == "")
```

```
    {
```

```
        BoolCheckArr[2] = false;
```

```
    }
```

```
    else {
```

```
        try {
```

```

        for (int i = 0; i < System::Convert::ToInt32(buf_str.length()); i++) {
            if (buf_str[i] == ',')
                buf_str[i] = '.';
        }

        changer_fix = gcnew System::String(buf_str.c_str());

        d = Double::Parse(GridTableRow_array[2]);

        if ((System::Convert::ToDouble(GridTableRow_array[2]) >= 2.0) &&
            (System::Convert::ToDouble(GridTableRow_array[2]) <= 5.0))

            BoolCheckArr[2] = true;
    }

    catch (...) {
        BoolCheckArr[2] = false;
    }
}

//Другие валидации
BoolCheckArr[3] = ((GridTableRow_array[3] == "да") || (GridTableRow_array[3] == "нет")) ?
true : false;

BoolCheckArr[4] = ((GridTableRow_array[4] == "очная") || (GridTableRow_array[4] ==
"заочная")) ? true : false;

BoolCheckArr[5] = ((GridTableRow_array[5] == "ПКС") || (GridTableRow_array[5] ==
"ИБАС")) ? true : false;

BoolCheckArr[6] = ((GridTableRow_array[6] == "оригинал") || (GridTableRow_array[6] ==
"копия")) ? true : false;

BoolCheckArr[7] = ((GridTableRow_array[7] == "бюджет") || (GridTableRow_array[7] ==
"договор")) ? true : false;

for (int i = 1; i < BoolCheckArr->Length; i++)
    if (BoolCheckArr[i] == false)
        return 1;

return 0;
}

private: DataTable^ GetDataTable() {
    DataTable ^table;

    SQLiteConnection ^db = gcnew SQLiteConnection();

```

```

db->ConnectionString = GlobalClass::SQLGlobalPatch;
db->Open();
SQLiteCommand ^cmdSelect = db->CreateCommand();
cmdSelect->CommandText = "SELECT * FROM students;";
SQLiteDataReader ^reader = cmdSelect->ExecuteReader();
DataColumn ^column;
DataRow ^row;
table = gcnew DataTable();

vector<String^>^ nameColumns = gcnew vector<String^>();
for (int i = 0; i < (reader->FieldCount); i++) {
    nameColumns->push_back(reader->GetName(i));
    column = gcnew DataColumn(nameColumns->at(i), String::typeid);
    table->Columns->Add(column);
}
while (reader->Read()) {
    row = table->NewRow();
    for (int i = 0; i < (reader->FieldCount); i++) {
        row[nameColumns->at(i)] = reader->GetValue(i)->ToString();
        reader->GetValue(i)->ToString();
    }
    table->Rows->Add(row);
}

db->Close();
return table;
}

private: System::Void ExitButton_Click(System::Object^ sender, System::EventArgs^ e) {
    this->Hide();
}

private: System::Void DBChangerForm_Load(System::Object^ sender, System::EventArgs^ e) {
    dataGridView1->DataSource = GetDataTable();
}

private: System::Void RemoveBDBButton_Click(System::Object^ sender, System::EventArgs^ e) {

```

```

int index = dataGridView1->CurrentCell->RowIndex;
String^ ID = dataGridView1->Rows[index]->Cells["ID"]->Value->ToString();
SQLiteConnection ^db = gcnew SQLiteConnection();
db->ConnectionString = GlobalClass::SQLGlobalPatch;
db->Open();
SQLiteCommand ^cmdInsertValue = db->CreateCommand();
cmdInsertValue->CommandText = "DELETE FROM students WHERE ID = " + ID + ";";
cmdInsertValue->ExecuteNonQuery();
db->Close();
dataGridView1->DataSource = GetDataTable();
MessageBox::Show("Успешное удаление данных");
}

private: System::Void AddDBButton_Click(System::Object^ sender, System::EventArgs^ e) {
    int index = dataGridView1->CurrentCell->RowIndex;
    GridTableRow_array = gcnew array<String^>(9);
    for (int i = 0; i < GridTableRow_array->Length; i++)
        GridTableRow_array[i] = dataGridView1->Rows[index]->Cells[i]->Value->ToString();

    if (valid_checker() == 0) {
        SQLiteConnection ^db = gcnew SQLiteConnection();
        db->ConnectionString = GlobalClass::SQLGlobalPatch;
        db->Open();
        SQLiteCommand ^cmdInsertValue = db->CreateCommand();
        cmdInsertValue->CommandText = "INSERT INTO students VALUES(NULL," +
GridTableRow_array[1] + "," + changer_fix +
        "," + GridTableRow_array[3] + "," + GridTableRow_array[4] + "," +
GridTableRow_array[5] +
        "," + GridTableRow_array[6] + "," + GridTableRow_array[7] + "," +
GridTableRow_array[8] + ");";
        cmdInsertValue->ExecuteNonQuery();
        db->Close();
        MessageBox::Show("Успешная запись данных");
    }
    else
        MessageBox::Show("Проверьте правильность ввода данных!");
    dataGridView1->DataSource = GetDataTable();
}

```

```

    }

private: System::Void UpdateBDButton_Click_1(System::Object^ sender, System::EventArgs^ e) {

    int index = dataGridView1->CurrentCell->RowIndex;

    String^ ID = dataGridView1->Rows[index]->Cells["ID"]->Value->ToString();

    GridTableRow_array = gcnew array<String^>(9);

    for (int i = 0; i < GridTableRow_array->Length; i++)

        GridTableRow_array[i] = dataGridView1->Rows[index]->Cells[i]->Value->ToString();

    if (valid_checker() == 0) {

        SQLiteConnection ^db = gcnew SQLiteConnection();

        db->ConnectionString = GlobalClass::SQLGlobalPatch;
        db->Open();
        SQLiteCommand ^cmdInsertValue = db->CreateCommand();
        cmdInsertValue->CommandText = "UPDATE students SET name=" +
GridTableRow_array[1] + ",score=" + changer_fix +

        ",priority=" + GridTableRow_array[3] + ",form_study=" +
GridTableRow_array[4] + ",major=" + GridTableRow_array[5] +

        ",original=" + GridTableRow_array[6] + ", form_pay=" +
GridTableRow_array[7] + ",date=" + GridTableRow_array[8] + " WHERE ID =" + ID;

        cmdInsertValue->ExecuteNonQuery();

        db->Close();

        MessageBox::Show("Успешное обновление данных");

    }

    else

        MessageBox::Show("Проверьте правильность ввода данных!");

    dataGridView1->DataSource = GetDataTable();

}

};

}

```

Модуль «FinalPrintForm.h»

#pragma endregion

```

array<String^>^ main_arr;

array<String^>^ StudContacts_arr;

```

```

array<String^>^ ParentsCont_arr;

public: String^ PublicStudentID;

private: void MainInfoGetter() {
    SQLiteConnection ^db = gcnew SQLiteConnection();
    main_arr = gcnew array<String^>(9);

    String^ SQL_STRING = (PublicStudentID == "0") ? "SELECT * FROM students WHERE ID = "
    (SELECT MAX(ID) FROM students);" : "SELECT * FROM students WHERE ID = " + PublicStudentID + ";";
    db->ConnectionString = GlobalClass::SQLGlobalPatch;

    db->Open();

    SQLiteCommand ^cmdSelect = db->CreateCommand();
    cmdSelect->CommandText = SQL_STRING;
    SQLiteDataReader ^data = cmdSelect->ExecuteReader();
    while (data->Read())
        for (int cell_index = 0; cell_index < data->FieldCount; cell_index++)
            main_arr[cell_index] = data->GetValue(cell_index)->ToString();
    db->Close();
}

private: void AllCheckBoxChecker() {
    MainInfoCheckBox->Checked = true;
    StudentContactsCheckBox->Checked = true;
    ParentsContactsCheckBox->Checked = true;
    DateCheckBox->Checked = true;
    TimeCheckBox->Checked = true;
    WriterCheckBox->Checked = true;
}

private: void NoneCheckBoxChecker() {
    MainInfoCheckBox->Checked = false;
    StudentContactsCheckBox->Checked = false;
    ParentsContactsCheckBox->Checked = false;
    DateCheckBox->Checked = false;
    TimeCheckBox->Checked = false;
}

```

```

        WriterCheckBox->Checked = false;
    }

private: void StudentContactsInfoGetter() {
    SQLiteConnection ^db = gcnew SQLiteConnection();
    StudContacts_arr = gcnew array<String^>(5);
    db->ConnectionString = GlobalClass::SQLGlobalPatch;

    String^ SQL_STRING = (PublicStudentID == "0") ? "SELECT * FROM contacts WHERE
student_id = (SELECT MAX(student_id) FROM contacts);" : "SELECT * FROM contacts WHERE student_id =
" + PublicStudentID + ";";

    db->Open();

    SQLiteCommand ^cmdSelect = db->CreateCommand();

    cmdSelect->CommandText = SQL_STRING;

    SQLiteDataReader ^data = cmdSelect->ExecuteReader();

    while (data->Read())
        for (int cell_index = 0; cell_index < data->FieldCount; cell_index++)
            StudContacts_arr[cell_index] = data->GetValue(cell_index)->ToString();

    db->Close();
}

private: void ParentsContactsInfoGetter() {
    SQLiteConnection ^db = gcnew SQLiteConnection();
    ParentsCont_arr = gcnew array<String^>(6);
    db->ConnectionString = GlobalClass::SQLGlobalPatch;

    String^ SQL_STRING = (PublicStudentID == "0") ? "SELECT * FROM parents WHERE
student_id = (SELECT MAX(student_id) FROM parents);" : "SELECT * FROM parents WHERE student_id =
" + PublicStudentID + ";";

    db->Open();

    SQLiteCommand ^cmdSelect = db->CreateCommand();

    cmdSelect->CommandText = SQL_STRING;

    SQLiteDataReader ^data = cmdSelect->ExecuteReader();

    while (data->Read())
        for (int cell_index = 0; cell_index < data->FieldCount; cell_index++)
            ParentsCont_arr[cell_index] = data->GetValue(cell_index)->ToString();

    db->Close();
}

private: void WordWorker() {

    System::DateTime now = System::DateTime::Now;

    String^ date_str = now.ToString("d");

    String^ time_str = now.ToString("t");
    Object ^ ФорматСтроки = Microsoft::Office::Interop::Word::WdUnits::wdLine;

```

```

auto t = Type::Missing;

auto WORD = gnew Microsoft::Office::Interop::Word::ApplicationClass();

WORD->Visible = true;

auto Документ = WORD->Documents->Add(t, t, t);

WORD->Selection->ParagraphFormat->Alignment =
    Microsoft::Office::Interop::Word::WdParagraphAlignment::wdAlignParagraphCenter;
//абзац по центру
WORD->Selection->Font->Name = ("Times New Roman"); //тип шрифта

WORD->Selection->Font->Bold = 1; // жирный шрифт

WORD->Selection->Font->Size = 18; // высота шрифта 18

WORD->Selection->TypeText("Справка о приеме документов в КИП при ФУ РФ");

WORD->Selection->Font->Size = 12;

WORD->Selection->TypeParagraph();

WORD->Selection->ParagraphFormat->Alignment =
Microsoft::Office::Interop::Word::WdParagraphAlignment::wdAlignParagraphJustify;

//Основная информация
if (MainInfoCheckBox->Checked == true) {
    MainInfoGetter();

    WORD->Selection->TypeParagraph();

    WORD->Selection->Font->Size = 16;

    WORD->Selection->TypeText("Основная информация:");

    WORD->Selection->Font->Bold = false;

    WORD->Selection->TypeParagraph();

    WORD->Selection->Font->Size = 14;

    WORD->Selection->TypeText(
        "ФИО: " + main_arr[1] + "\n" +
        "Средний балл аттестата: " + main_arr[2] + "\n" +
        "Направление: " + main_arr[5] + "\n" +
        "Форма оплаты: " + main_arr[7] + "\n" +
        "Тип аттестата: " + main_arr[6] + "\n" +
        "Форма обучения: " + main_arr[4] + "\n" +
        "Приоритет: " + main_arr[3] + "\n"
    );
}

```



```

if (StudentContactsCheckBox->Checked == true) {
    StudentContactsInfoGetter();

    WORD->Selection->TypeParagraph();

    WORD->Selection->Font->Size = 16;

    WORD->Selection->Font->Bold = true;

    WORD->Selection->TypeText("Контактные данные абитуриента:");

    WORD->Selection->Font->Bold = false;

    WORD->Selection->TypeParagraph();

    WORD->Selection->Font->Size = 14;

    WORD->Selection->TypeText(
        "Адрес: " + StudContacts_arr[2] + "\n" +
        "E-mail: " + StudContacts_arr[3] + "\n" +
        "Телефон: " + StudContacts_arr[4] + "\n"
    );
}

if (ParentsContactsCheckBox->Checked == true) {
    ParentsContactsInfoGetter();

    WORD->Selection->TypeParagraph();

    WORD->Selection->Font->Size = 16;

    WORD->Selection->Font->Bold = true;

    WORD->Selection->TypeText("Контактные данные родителей абитуриента:");

    WORD->Selection->Font->Bold = false;

    WORD->Selection->TypeParagraph();

    WORD->Selection->Font->Size = 14;

    WORD->Selection->TypeText(
        "ФИО: " + ParentsCont_arr[2] + "\n" +
        "Адрес: " + ParentsCont_arr[4] + "\n" +
        "E-mail: " + ParentsCont_arr[3] + "\n" +
        "Телефон: " + ParentsCont_arr[5] + "\n"
    );
}

if (DateCheckBox->Checked == true) {
    WORD->Selection->ParagraphFormat->Alignment =
Microsoft::Office::Interop::Word::WdParagraphAlignment::wdAlignParagraphRight;
    WORD->Selection->TypeText("Дата: " + date_str + "\n");
}

```

```

    }

    if (TimeCheckBox->Checked == true) {
        WORD->Selection->ParagraphFormat->Alignment =
Microsoft::Office::Interop::Word::WdParagraphAlignment::wdAlignParagraphRight;
        WORD->Selection->TypeText("Время: " + time_str + "\n");
    }

    if (WriterCheckBox->Checked == true) {
        WORD->Selection->ParagraphFormat->Alignment =
Microsoft::Office::Interop::Word::WdParagraphAlignment::wdAlignParagraphRight;
        WORD->Selection->TypeText(
            "Подпись: _____\n" +
            "Расшифровка: _____"
        );
    }
}

private: System::Void ExitButton_Click(System::Object^ sender, System::EventArgs^ e) {
    this->Hide();
}

private: System::Void ReportButton_Click(System::Object^ sender, System::EventArgs^ e) {
    WordWorker();
}

private: System::Void AllCheckBox_Click(System::Object^ sender, System::EventArgs^ e) {
    AllCheckBoxChecker();
}

private: System::Void NoneCheckBox_Click(System::Object^ sender, System::EventArgs^ e) {
    NoneCheckBoxChecker();
}

};
}

```

Модуль «InputForm.h»

#pragma endregion

String^ changer_fix;

```

String^ FIO_public;

array<bool>^ valid_array;

private: int valid_checker(){

    bool main_checker = true;

    for (int i = 0; i < valid_array->Length; i++)

        if (valid_array[i] == false)

            main_checker = false;

    if (

        (main_checker == true)

        &&

        (OriginalRadioButton1->Checked == true || OriginalRadioButton2->Checked == true)

        &&

        (StudyformRadioButton1->Checked == true || StudyformRadioButton2->Checked ==

true)

        &&

        (PriorityRadioButton1->Checked == true || PriorityRadioButton2->Checked == true)

        &&

        (FormPayRadioButton1->Checked == true || FormPayRadioButton2->Checked == true)

    )

        return 1;

    return 0;

}

private: void print_outer() {

    if (valid_checker() == 1) {

        Object ^ ФорматСтроки = Microsoft::Office::Interop::Word::WdUnits::wdLine;

        auto t = Type::Missing;

        String^ original_str = (OriginalRadioButton1->Checked == true) ? "оригинал" :

"копия";

        String^ priority_str = (PriorityRadioButton1->Checked == true) ? "есть" : "нет";

        String^ form_study_str = (StudyformRadioButton1->Checked == true) ? "очная" :

"заочная";

        String^ form_pay_str = (FormPayRadioButton1->Checked == true) ? "бюджет" :

"договор";

        System::DateTime now = System::DateTime::Now;

        String^ date_str = now.ToString("d");

        String^ time_str = now.ToString("t");

```

```

try
{
    auto WORD = gcnew Microsoft::Office::Interop::Word::ApplicationClass();

    //Делаем видимым все происходящее
    WORD->Visible = true;

    // Открываем новый документ
    auto Документ = WORD->Documents->Add(t, t, t);

    WORD->Selection->ParagraphFormat->Alignment =

Microsoft::Office::Interop::Word::WdParagraphAlignment::wdAlignParagraphCenter; //абзац по центру

    WORD->Selection->Font->Name = ("Times New Roman"); //тип шрифта
    WORD->Selection->Font->Bold = 1; // жирный шрифт
    WORD->Selection->Font->Size = 18; // высота шрифта 18
    WORD->Selection->TypeText("Справка о приеме документов в КИП при
ФУ РФ");

    WORD->Selection->Font->Size = 12;
    WORD->Selection->TypeParagraph();
    WORD->Selection->ParagraphFormat->Alignment =

Microsoft::Office::Interop::Word::WdParagraphAlignment::wdAlignParagraphJustify;

    WORD->Selection->TypeParagraph();
    WORD->Selection->Font->Size = 16;
    WORD->Selection->TypeText("Основная информация:");
    WORD->Selection->Font->Bold = false;
    WORD->Selection->TypeParagraph();
    WORD->Selection->Font->Size = 14;
    WORD->Selection->TypeText(
        "ФИО: " + FIOBox->Text + "\n" +
        "Средний балл аттестата: " + ScoreBox->Text + "\n" +
        "Направление: " + MajorComboBox->Text + "\n"
    );

    WORD->Selection->TypeParagraph();

```

```

WORD->Selection->Font->Bold = true;

WORD->Selection->Font->Size = 16; // высота шрифта 16

WORD->Selection->TypeText("Дополнительная информация:");

WORD->Selection->Font->Bold = false;

WORD->Selection->TypeParagraph();

WORD->Selection->Font->Size = 14;

WORD->Selection->TypeText(
    "Форма оплаты: " + form_pay_str + "\n" +
    "Тип аттестата: " + original_str + "\n" +
    "Форма обучения: " + form_study_str + "\n" +
    "Приоритет: " + priority_str + "\n"
);

WORD->Selection->TypeParagraph();

WORD->Selection->ParagraphFormat->Alignment =

Microsoft::Office::Interop::Word::WdParagraphAlignment::wdAlignParagraphRight;

WORD->Selection->TypeText(
    "Дата: " + date_str + "\n" +
    "Время: " + time_str + "\n" +
    "Подпись: _____\n" +
    "Расшифровка: _____"
);
}
catch (Exception^ ex)
{
    MessageBox::Show(ex->Message, "Ошибка", MessageBoxButtons::OK,
        MessageBoxIcon::Exclamation);
}
}
else
    MessageBox::Show("Проверьте правильность ввода данных!");
}

private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e) {

```

```

if (valid_checker() == 1) {
    System::DateTime now = System::DateTime::Now;
    String^ date_str = now.ToString("d");
    String ^priority_str, ^form_study_str, ^original_str, ^form_pay_str;
    SQLiteConnection ^db = gcnew SQLiteConnection();
    original_str = (OriginalRadioButton1->Checked == true) ? "оригинал" : "копия";
    priority_str = (PriorityRadioButton1->Checked == true) ? "да" : "нет";
    form_study_str = (StudyformRadioButton1->Checked == true) ? "очная" : "заочная";
    form_pay_str = (FormPayRadioButton1->Checked == true) ? "бюджет" : "договор";

    try
    {
        db->ConnectionString = GlobalClass::SQLGlobalPatch;
        db->Open();
        SQLiteCommand ^cmdInsertValue = db->CreateCommand();
        cmdInsertValue->CommandText = "CREATE TABLE IF NOT
EXISTS students" +
"(ID INTEGER PRIMARY KEY AUTOINCREMENT NOT
NULL, name TEXT, score REAL, priority TEXT, form_study TEXT, major TEXT, original TEXT, form_pay
TEXT, date TEXT);" +
"INSERT INTO students VALUES(NULL,'" + FIOBox->Text
+ "','" + changer_fix +
MajorComboBox->Text +
,'" + priority_str + "','" + form_study_str + "','" +
,'" + original_str + "','" + form_pay_str + "','" + date_str +
");";

        cmdInsertValue->ExecuteNonQuery();
        db->Close();
    }
    finally
    {
        delete (IDisposable^)db;
    }

    if (MessageBox::Show("Данные успешно записаны!\nХотите ввести
контактные данные абитуриента?", "Ввод контактных данных",
System::Windows::Forms::MessageBoxButtons::YesNo) == System::Windows::Forms::DialogResult::Yes)
    {

```

```

        DEMKA::MoreStudentInfoForm^MoreStudentInfoForm_obj = gnew
DEMKA::MoreStudentInfoForm();

        this->Hide();

        MoreStudentInfoForm_obj->Text = FIO_public;

        MoreStudentInfoForm_obj->ShowDialog();

    }

}

else

    MessageBox::Show("Проверьте правильность ввода данных!");

}

private: System::Void MenuButton_Click(System::Object^ sender, System::EventArgs^ e) {

    this->Hide();

}

private: System::Void InputForm_Load(System::Object^ sender, System::EventArgs^ e) {

    valid_array = gnew array<bool>(3);
    for (int i = 0; i < valid_array->Length; i++)
        valid_array[i] = false;

    MajorComboBox->DropDownStyle = ComboBoxStyle::DropDownList;

    MajorComboBox->Items->Add("ИКС");

    MajorComboBox->Items->Add("ИБАС");

}

//Вывод на печать

private: System::Void PrinterButton_Click(System::Object^ sender, System::EventArgs^ e) {

    print_outer();

}

private: System::Void ScoreBox_TextChanged(System::Object^ sender, System::EventArgs^ e) {

    msclr::interop::marshal_context context;

    std::string buf_str = context.marshal_as<std::string>(ScoreBox->Text);

    double d;

    if (ScoreBox->Text == "")

    {

        valid_array[1] = false;

    }

    else {

```

```

try {
    for (int i = 0; i < System::Convert::ToInt32(buf_str.length()); i++) {
        if (buf_str[i] == ',')
            buf_str[i] = '.';
    }

    changer_fix = gnew System::String(buf_str.c_str());
    d = Double::Parse(ScoreBox->Text);

    if ((System::Convert::ToDouble(ScoreBox->Text) >= 2.0) &&
        (System::Convert::ToDouble(ScoreBox->Text) <= 5.0)) {

        ScoreBox->ForeColor =
System::Drawing::SystemColors::WindowText;

        valid_array[1] = true;
    }
    else
        throw e;
}
catch (...) {
    valid_array[1] = false;
    ScoreBox->ForeColor = System::Drawing::Color::Red;
}
}

private: System::Void FIOBox_TextChanged(System::Object^ sender, System::EventArgs^ e) {

```

```

    if (FIOBox->Text == "")
        valid_array[0] = false;
    else {
        valid_array[0] = true;
        FIO_public = FIOBox->Text;
        msclr::interop::marshal_context oMarshalContext;
        const char* buf = oMarshalContext.marshal_as<const char*>(FIOBox->Text);
        for (int i = 0; i < System::Convert::ToInt32(strlen(buf)); i++)
            if (iswdigit(buf[i]))
                valid_array[0] = false;
    }
}

```



```

        FIOBox->ForeColor = (valid_array[0] == true) ?
System::Drawing::SystemColors::WindowText : System::Drawing::Color::Red;

    }

}

private: System::Void MajorComboBox_TextChanged(System::Object^ sender, System::EventArgs^ e)
{
    if (MajorComboBox->Text == "")
        valid_array[2] = false;

    else
        valid_array[2] = true;
}

};

}

```

Модуль «MainDBChangerForm.h»

#pragma endregion

```

private: System::Void ExitButton_Click(System::Object^ sender, System::EventArgs^ e) {
    this->Hide();
}

private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e) {
    this->Hide();
}

private: System::Void StudentsTableButton_Click(System::Object^ sender, System::EventArgs^ e) {
    DEMKA::DBChangerForm^DBChangerForm_obj = gcnew DEMKA::DBChangerForm();

    this->Hide();

    DBChangerForm_obj->ShowDialog();

    this->Show();
}

private: System::Void ContactsTableButton_Click(System::Object^ sender, System::EventArgs^ e) {
    DEMKA::ContactsDBChangerForm^ContactsDBChangerForm_obj = gcnew
DEMKA::ContactsDBChangerForm();

    this->Hide();

    ContactsDBChangerForm_obj->ShowDialog();

    this->Show();
}

private: System::Void ParentsTableButton_Click(System::Object^ sender, System::EventArgs^ e) {
    DEMKA::ParentsDBChangerForm^ParentsDBChangerForm_obj = gcnew
DEMKA::ParentsDBChangerForm();

    this->Hide();

    ParentsDBChangerForm_obj->ShowDialog();

    this->Show();
}

```

```

private: System::Void StaffManagmentButton_Click(System::Object^ sender, System::EventArgs^ e) {
    DEMKA::MainStaffForm^MainStaffForm_obj = gnew DEMKA::MainStaffForm();

    this->Hide();

    MainStaffForm_obj->ShowDialog();

    this->Show();

}

};

}

Модуль «MainForm.h»

#pragma endregion

private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e) {

    InputForm^InputForm_obj = gnew InputForm();

    this->Hide();

    InputForm_obj->ShowDialog();

    this->Show();

}

private: System::Void ExitButton_Click(System::Object^ sender, System::EventArgs^ e) {

    if (MessageBox::Show("Вы действительно хотите выйти из системы?", "Выход из АИС",
        MessageBoxButtons::YesNo, MessageBoxIcon::Question) == System::Windows::Forms::DialogResult::Yes)

        this->Hide();

}

private: System::Void TopButton_Click(System::Object^ sender, System::EventArgs^ e) {

    DEMKA::RatingForm^RatingForm_obj = gnew DEMKA::RatingForm();

    this->Hide();

    RatingForm_obj->ShowDialog();

    this->Show();

}

private: System::Void button2_Click(System::Object^ sender, System::EventArgs^ e) {

    DEMKA::PasswordChecker^PasswordChecker_obj = gnew DEMKA::PasswordChecker();

    this->Hide();

    PasswordChecker_obj->ShowDialog();

    this->Show();

}

};

```

```
}
```

Модуль «MainStaffForm.h»

```
#pragma endregion
```

```
private: System::Void MenuButton_Click(System::Object^ sender, System::EventArgs^ e) {
    this->Hide();
}
private: System::Void AddNewStaffButton_Click(System::Object^ sender, System::EventArgs^ e) {
    DEMKA::AddNewStaffForm^AddNewStaffForm_obj = gcnew DEMKA::AddNewStaffForm();
    this->Hide();
    AddNewStaffForm_obj->ShowDialog();
    this->Show();
}
private: System::Void PasswordStaffButton_Click(System::Object^ sender, System::EventArgs^ e) {
    DEMKA::PasswordStaffForm^PasswordStaffForm_obj = gcnew
DEMKA::PasswordStaffForm();
    this->Hide();
    PasswordStaffForm_obj->ShowDialog();
    this->Show();
}
private: System::Void RemoveStaffButton_Click(System::Object^ sender, System::EventArgs^ e) {
    DEMKA::RemoveStaffForm^RemoveStaffForm_obj = gcnew DEMKA::RemoveStaffForm();
    this->Hide();
    RemoveStaffForm_obj->ShowDialog();
    this->Show();
}
};
}
```

Модуль «MoreStudentInfoForm.h»

```
#pragma endregion
```

```
array<bool>^ valid_array;
private: int ID_getter() {
    SQLiteConnection ^db = gcnew SQLiteConnection();
    String^ columnIndex;
    db->ConnectionString = GlobalClass::SQLGlobalPatch;
    db->Open();
    SQLiteCommand ^cmdSelect = db->CreateCommand();
    cmdSelect->CommandText = "SELECT * FROM students WHERE ID = (SELECT MAX(ID)
FROM students);";
    SQLiteDataReader ^data = cmdSelect->ExecuteReader();
}
```

```

while (data->Read())
{
    columnIndex = data->GetValue(0)->ToString();
}

db->Close();

return System::Convert::ToInt32(columnIndex);

}

private: int valid_checker() {
    for (int i = 0; i < valid_array->Length; i++)
        if (valid_array[i] == false)
            return 1;
    return 0;
}

private: System::Void NextButton_Click(System::Object^ sender, System::EventArgs^ e) {
    if (valid_checker() == 0) {
        SQLiteConnection ^db = gcnew SQLiteConnection();
        try
        {
            db->ConnectionString = GlobalClass::SQLGlobalPatch;

            db->Open();

            SQLiteCommand ^cmdInsertValue = db->CreateCommand();

            cmdInsertValue->CommandText =

                "CREATE TABLE IF NOT EXISTS contacts(" +

                "ID INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL," +
                "student_id INTEGER UNIQUE NOT NULL," +
                "adress TEXT," +
                "e_mail TEXT," +
                "telephone TEXT," +
                "FOREIGN KEY(student_id) REFERENCES students(ID) ON
DELETE CASCADE);" +

                "INSERT INTO contacts VALUES(NULL," +
                System::Convert::ToString(ID_getter()) + ", " + AddressBox->Text + ", " + EmailBox->Text + ", " + PhoneBox->Text + "));";

            cmdInsertValue->ExecuteNonQuery();

            db->Close();

        }
    }
}

```

```

        finally
        {
            delete (IDisposable^)db;
        }
        if (MessageBox::Show("Контактные данные успешно записаны!\nХотите ввести
контактные данные родителей?", "Контактные данные родителей",
System::Windows::Forms::MessageBoxButtons::YesNo) == System::Windows::Forms::DialogResult::Yes)
        {
            DEMKA::ParentsContactForm^ParentsContactForm_obj = gnew
DEMKA::ParentsContactForm();
            this->Hide();
            ParentsContactForm_obj->Text = this->Text;
            ParentsContactForm_obj->StudentAdress = AdressBox->Text;
            ParentsContactForm_obj->ShowDialog();
        }
        else
            this->Hide();
    }
    else
        MessageBox::Show("Проверьте правильность ввода данных!");
}

private: System::Void MenuButton_Click(System::Object^ sender, System::EventArgs^ e) {
    this->Hide();
}

private: System::Void MoreStudentInfoForm_Load(System::Object^ sender, System::EventArgs^ e) {
    valid_array = gnew array<bool>(3);
    for (int i = 0; i < valid_array->Length; i++)
        valid_array[i] = false;
}

private: System::Void AdressBox_TextChanged(System::Object^ sender, System::EventArgs^ e) {
    valid_array[0] = (AdressBox->Text != "") ? true : false;
}

private: System::Void EmailBox_TextChanged(System::Object^ sender, System::EventArgs^ e) {
    System::String^ buf_str = EmailBox->Text;
    std::string Email_str = msclr::interop::marshal_as<std::string>(buf_str);

```

```

        valid_array[1] = (GlobalClass::is_valid_email(Email_str)) ? true : false;

        EmailBox->ForeColor = GlobalClass::is_valid_email(Email_str) ?
System::Drawing::SystemColors::WindowText : System::Drawing::Color::Red;

    }

    private: System::Void PhoneBox_TextChanged(System::Object^ sender, System::EventArgs^ e) {

        System::String^ buf_str = PhoneBox->Text;

        std::string Phone_str = msclr::interop::marshal_as<std::string>(buf_str);

        bool boolflag = GlobalClass::is_valid_number(Phone_str) && Phone_str.size() == 11 &&
Phone_str[0] == '7';

        valid_array[2] = (boolflag) ? true : false;

        PhoneBox->ForeColor = (boolflag) ? System::Drawing::SystemColors::WindowText :
System::Drawing::Color::Red;

    }

};

}

```

Модуль «ParentsContactForm.h»

#pragma endregion

```

    public: String^ StudentAdress;
    private: array<bool>^ valid_array;
    private: int ID_getter() {
        SQLiteConnection ^db = gcnew SQLiteConnection();
        String^ columnIndex;

        db->ConnectionString = GlobalClass::SQLGlobalPatch;

        db->Open();

        SQLiteCommand ^cmdSelect = db->CreateCommand();

        cmdSelect->CommandText = "SELECT * FROM students WHERE ID = (SELECT MAX(ID)
FROM students);";

        SQLiteDataReader ^data = cmdSelect->ExecuteReader();

        while (data->Read())
        {

            columnIndex = data->GetValue(0)->ToString();

        }

        db->Close();

        return System::Convert::ToInt32(columnIndex);

    }

    private: int valid_checker() {

```

```

        for (int i = 0; i < valid_array->Length; i++)
            if (valid_array[i] == false)
                return 1;

        return 0;
    }

    private: System::Void label4_Click(System::Object^ sender, System::EventArgs^ e) {
    }

    private: System::Void ExitButton_Click(System::Object^ sender, System::EventArgs^ e) {
        this->Hide();
    }

    private: System::Void NextButton_Click(System::Object^ sender, System::EventArgs^ e) {
        if (valid_checker() == 0) {
            SQLiteConnection ^db = gcnew SQLiteConnection();
            try
            {
                db->ConnectionString = GlobalClass::SQLGlobalPatch;

                db->Open();

                SQLiteCommand ^cmdInsertValue = db->CreateCommand();

                cmdInsertValue->CommandText =

                    "CREATE TABLE IF NOT EXISTS parents(" +

                    "ID INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL," +

                    "student_id INTEGER UNIQUE NOT NULL," +
                    "name TEXT," +
                    "e_mail TEXT," +
                    "adress TEXT," +
                    "telephone TEXT," +
                    "FOREIGN KEY(student_id) REFERENCES students(ID) ON
DELETE CASCADE);" +

                    "INSERT INTO parents VALUES (NULL," +
                    System::Convert::ToString(ID_getter()) + ", '" + ParentFIOBox->Text + "', '" + ParentEmailBox->Text + "', '" +
                    ParentAdressBox->Text + "', '" + ParentPhoneBox->Text + "');"

                cmdInsertValue->ExecuteNonQuery();

                db->Close();
            }
            finally
            {
                delete (IDisposable^)db;
            }
        }
    }

```

```

        }
        if (MessageBox::Show("Контактные данные родителей успешно записаны!\nХотите
сформировать общий отчет?", "Формирование отчета",
System::Windows::Forms::MessageBoxButtons::YesNo) == System::Windows::Forms::DialogResult::Yes)
    {
        DEMKA::FinalPrintForm^FinalPrintForm_obj = gcnew
DEMKA::FinalPrintForm();

        FinalPrintForm_obj->PublicStudentID = "0";

        this->Hide();

        FinalPrintForm_obj->ShowDialog();

    }
    else
        this->Hide();
    }
    else
        MessageBox::Show("Проверьте правильность ввода данных!");
    }
private: System::Void ParentFIOBox_TextChanged(System::Object^ sender, System::EventArgs^ e) {
    if (ParentFIOBox->Text == "")
    {
        valid_array[0] = false;
    }
    else {
        valid_array[0] = true;

        msclr::interop::marshal_context oMarshalContext;

        const char* buf = oMarshalContext.marshal_as<const char*>(ParentFIOBox->Text);

        for (int i = 0; i < System::Convert::ToInt32(strlen(buf)); i++) {
            if (iswdigit(buf[i])) {
                valid_array[0] = false;
            }
        }

        ParentFIOBox->ForeColor = (valid_array[0] == true) ?
System::Drawing::SystemColors::WindowText : System::Drawing::Color::Red;
    }
}

private: System::Void ParentsContactForm_Load(System::Object^ sender, System::EventArgs^ e) {
    valid_array = gcnew array<bool>(4);

    for (int i = 0; i < valid_array->Length; i++)

        valid_array[i] = false;
}

```



```

    }

private: System::Void ParentAdressBox_TextChanged(System::Object^ sender, System::EventArgs^ e)
{
    valid_array[1] = (ParentAdressBox->Text=="") ? false : true;
}

private: System::Void ParentEmailBox_TextChanged(System::Object^ sender, System::EventArgs^ e) {
    System::String^ buf_str = ParentEmailBox->Text;
    std::string Email_str = msclr::interop::marshal_as<std::string>(buf_str);
    valid_array[2] = (GlobalClass::is_valid_email(Email_str)) ? true : false;
    ParentEmailBox->ForeColor = GlobalClass::is_valid_email(Email_str) ?
System::Drawing::SystemColors::WindowText : System::Drawing::Color::Red;
}

private: System::Void ParentPhoneBox_TextChanged(System::Object^ sender, System::EventArgs^ e) {
    System::String^ buf_str = ParentPhoneBox->Text;
    std::string Phone_str = msclr::interop::marshal_as<std::string>(buf_str);
    bool boolflag = GlobalClass::is_valid_number(Phone_str) && Phone_str.size() == 11 &&
Phone_str[0] == '7';
    valid_array[3] = (boolflag) ? true : false;
    ParentPhoneBox->ForeColor = (boolflag) ? System::Drawing::SystemColors::WindowText :
System::Drawing::Color::Red;
}

private: System::Void AdressBox_CheckedChanged(System::Object^ sender, System::EventArgs^ e) {
    if (AdressBox->Checked) {
        ParentAdressBox->Enabled = false;
        ParentAdressBox->Text = StudentAddress;
    }
    else {
        ParentAdressBox->Enabled = true;
        ParentAdressBox->Text = "";
    }
}

};
}

```

#pragma endregion

```
String^ changer_fix;

array<String^>^ GridTableRow_array;

private: int valid_checker() {

    array<bool>^ BoolCheckArr;
    BoolCheckArr = gcnew array<bool>(6);
    for (int i = 0; i < BoolCheckArr->Length; i++) {
        BoolCheckArr[i] = false;
    }

    //Валидация e-mail
    if (GridTableRow_array[3] == "")
    {
        BoolCheckArr[3] = false;
    }
    else {
        BoolCheckArr[3] = true;
        msclr::interop::marshal_context oMarshalContext;
        const char* buf = oMarshalContext.marshal_as<const
char*>(GridTableRow_array[3]);
        if (GlobalClass::is_valid_email(buf) == false)
            BoolCheckArr[3] = false;
    }

    //Валидация телефона
    if (GridTableRow_array[5] == "")
        BoolCheckArr[5] = false;
    else{
        BoolCheckArr[5] = true;
        msclr::interop::marshal_context oMarshalContext;
        const char* buf = oMarshalContext.marshal_as<const
char*>(GridTableRow_array[5]);
        if (GlobalClass::is_valid_number(buf) == false)
            BoolCheckArr[5] = false;
    }
}
```

```

    }

    //Валидация ФИО
    if (GridTableRow_array[2] == "")
        BoolCheckArr[2] = false;
    else {
        BoolCheckArr[2] = true;
        msclr::interop::marshal_context oMarshalContext;
        const char* buf_char = oMarshalContext.marshal_as<const
char*>(GridTableRow_array[2]);
        for (int i = 0; i < System::Convert::ToInt32(strlen(buf_char)); i++)
            if (iswdigit(buf_char[i]))
                BoolCheckArr[2] = false;
    }

    //Ждем все
    BoolCheckArr[4] = true;
    for (int i = 2; i < BoolCheckArr->Length; i++)
        if (BoolCheckArr[i] == false)
            return 1;
    return 0;
}

```

```

private: DataTable^ GetDataTable() {
    DataTable ^table;
    SQLiteConnection ^db = gcnew SQLiteConnection();

    db->ConnectionString = GlobalClass::SQLGlobalPatch;
    db->Open();
    SQLiteCommand ^cmdSelect = db->CreateCommand();
    cmdSelect->CommandText = "SELECT * FROM parents;";
    SQLiteDataReader ^reader = cmdSelect->ExecuteReader();
    DataColumn ^column;
    DataRow ^row;
}

```

```

        table = gcnew DataTable();
        vector<String^> nameColumns = gcnew vector<String^>();

        for (int i = 0; i < (reader->FieldCount); i++) {
            nameColumns->push_back(reader->GetName(i));
            column = gcnew DataColumn(nameColumns->at(i), String::typeid);
            table->Columns->Add(column);
        }

        while (reader->Read()) {
            row = table->NewRow();
            for (int i = 0; i < (reader->FieldCount); i++) {
                row[nameColumns->at(i)] = reader->GetValue(i)->ToString();
                reader->GetValue(i)->ToString();
            }
            table->Rows->Add(row);
        }
        db->Close();
        return table;
    }

```

```

private: System::Void ExitButton_Click(System::Object^ sender, System::EventArgs^ e) {
    this->Hide();
}

```

```

private: System::Void ParentsDBChangerForm_Load(System::Object^ sender, System::EventArgs^ e) {
    dataGridView1->DataSource = GetDataTable();
}

```

```

private: System::Void RemoveBDBButton_Click(System::Object^ sender, System::EventArgs^ e) {
    int index = dataGridView1->CurrentCell->RowIndex;
    String^ ID = dataGridView1->Rows[index]->Cells["ID"]->Value->ToString();

    SQLiteConnection ^db = gcnew SQLiteConnection();
    db->ConnectionString = GlobalClass::SQLGlobalPatch;
}

```

```

db->Open();

SQLiteCommand ^cmdInsertValue = db->CreateCommand();

cmdInsertValue->CommandText = "DELETE FROM parents WHERE ID = " + ID + ";";

cmdInsertValue->ExecuteNonQuery();

db->Close();

dataGridView1->DataSource = GetDataTable();

MessageBox::Show("Успешное удаление данных");

}

private: System::Void AddDBButton_Click(System::Object^ sender, System::EventArgs^ e) {

    int index = dataGridView1->CurrentCell->RowIndex;

    GridTableRow_array = gcnew array<String^>(6);

    for (int i = 0; i < GridTableRow_array->Length; i++)

        GridTableRow_array[i] = dataGridView1->Rows[index]->Cells[i]->Value->ToString();

    if (valid_checker() == 0) {

        SQLiteConnection ^db = gcnew SQLiteConnection();

        db->ConnectionString = GlobalClass::SQLGlobalPatch;

        db->Open();

        SQLiteCommand ^cmdInsertValue = db->CreateCommand();

        cmdInsertValue->CommandText = "INSERT INTO parents VALUES(NULL," +
GridTableRow_array[1] + "," + GridTableRow_array[2] +
        "," + GridTableRow_array[3] + "," + GridTableRow_array[4] + "," +
GridTableRow_array[5] + ")";

        cmdInsertValue->ExecuteNonQuery();

        db->Close();

        MessageBox::Show("Успешная запись данных");

    }

    else

        MessageBox::Show("Проверьте правильность ввода данных!");

    dataGridView1->DataSource = GetDataTable();

}

private: System::Void UpdateBDBButton_Click(System::Object^ sender, System::EventArgs^ e) {

```

```

int index = dataGridView1->CurrentCell->RowIndex;

String^ ID = dataGridView1->Rows[index]->Cells["ID"]->Value->ToString();

GridTableRow_array = gcnew array<String^>(6);

for (int i = 0; i < GridTableRow_array->Length; i++)

    GridTableRow_array[i] = dataGridView1->Rows[index]->Cells[i]->Value->ToString();

if (valid_checker() == 0) {

    SQLiteConnection ^db = gcnew SQLiteConnection();

    db->ConnectionString = GlobalClass::SQLGlobalPatch;
    db->Open();

    SQLiteCommand ^cmdInsertValue = db->CreateCommand();
    cmdInsertValue->CommandText = "UPDATE parents SET student_id=" +
GridTableRow_array[1] + ",name=" + GridTableRow_array[2] +
        ",e_mail=" + GridTableRow_array[3] + ",adress=" + GridTableRow_array[4]
+ ",telephone=" + GridTableRow_array[5] + " WHERE ID =" + ID;

    cmdInsertValue->ExecuteNonQuery();

    db->Close();

    MessageBox::Show("Успешное обновление данных");

}

else

    MessageBox::Show("Проверьте правильность ввода данных!");

dataGridView1->DataSource = GetDataTable();

}

};

}

```

Модуль «PasswordChecker.h»

```
#pragma endregion
```

```

private: bool PasswordValidation(String^ input_str) {

    if (GlobalClass::getMD5String(GlobalClass::toBase64(input_str)) ==
GlobalClass::MasterGlobalPassword)

        return true;

    return false;
}

```

```

    }

    private: System::Void PasswordCheckBox_TextChanged(System::Object^ sender,
System::EventArgs^ e) {

    }

    private: System::Void NextButton_Click(System::Object^ sender, System::EventArgs^ e) {

        if (PasswordValidation(PasswordCheckBox->Text)) {
            MessageBox::Show("Успешная авторизация!");
            DEMKA::MainDBChangerForm^MainDBChangerForm_obj = gcnew
DEMKA::MainDBChangerForm();

            this->Hide();

            MainDBChangerForm_obj->ShowDialog();

        }

        else

            MessageBox::Show("Неверный пароль, попробуйте ввести еще раз");

    }

    private: System::Void ExitButton_Click(System::Object^ sender, System::EventArgs^ e) {

        this->Hide();

    }

};

}

```

Модуль «PasswordStaffForm.h»

#pragma endregion

```

private: void EmptyAllTextBox() {

    StaffLoginBox->Text = "";

    StaffOldPasswordBox->Text = "";

    StaffNewPasswordBox->Text = "";

}

private: bool StaffLoginSQLChecker(String^ login_str, String^ password_str) {

    SQLiteConnection ^db = gcnew SQLiteConnection();

    String^ BufChecker = "DEMKA";

    String^ MD5login_str = GlobalClass::getMD5String(GlobalClass::toBase64(login_str));

    String^ MD5password_str = GlobalClass::getMD5String(GlobalClass::toBase64(password_str));

```

```

        String^ SQL_STRING = "SELECT * FROM staff WHERE login ='" + MD5login_str + "' AND
password ='" + MD5password_str + "'";
        db->ConnectionString = GlobalClass::SQLGlobalPatch;

        db->Open();

        SQLiteCommand ^cmdSelect = db->CreateCommand();

        cmdSelect->CommandText = SQL_STRING;

        SQLiteDataReader ^data = cmdSelect->ExecuteReader();

        while (data->Read())
            BufChecker = data->GetValue(0)->ToString();

        db->Close();

        if (BufChecker != "DEMKA")

            return true;

        return false;

    }

private: void UpdateStaffPasswordSQL(String^ login_str, String^ password_str) {

    SQLiteConnection ^db = gcnew SQLiteConnection();

    String^ MD5login_str = GlobalClass::getMD5String(GlobalClass::toBase64(login_str));

    String^ MD5NewPassword_str =
GlobalClass::getMD5String(GlobalClass::toBase64(password_str));

    String^ SQL_STRING = "UPDATE staff SET password= '" + MD5NewPassword_str + "'
WHERE login = '" + MD5login_str+"';";

    db->ConnectionString = GlobalClass::SQLGlobalPatch;

    db->Open();

    SQLiteCommand ^cmdInsertValue = db->CreateCommand();
    cmdInsertValue->CommandText = SQL_STRING;

    cmdInsertValue->ExecuteNonQuery();

    db->Close();

    MessageBox::Show("Успешное обновление пароля пользователя " + login_str + " !");

    this->Hide();

}

private: bool StaffExistLoginSQLChecker(String^ login_str) {

    SQLiteConnection ^db = gcnew SQLiteConnection();

```



```

String^ MD5login_str = GlobalClass::getMD5String(GlobalClass::toBase64(login_str));

String^ BufValidationStr = "DEMKA";
String^ SQL_STRING = "SELECT * FROM staff WHERE login =" + MD5login_str + """;

db->ConnectionString = GlobalClass::SQLGlobalPatch;

db->Open();

SQLiteCommand ^cmdSelect = db->CreateCommand();

cmdSelect->CommandText = SQL_STRING;

SQLiteDataReader ^data = cmdSelect->ExecuteReader();


while (data->Read())
    BufValidationStr = data->GetValue(0)->ToString();

db->Close();


if (BufValidationStr != "DEMKA")
    return true;

return false;
}

private: System::Void ExitButton_Click(System::Object^ sender, System::EventArgs^ e) {
    this->Hide();
}

private: System::Void ChangePasswordButton_Click(System::Object^ sender, System::EventArgs^ e) {
    if (StaffExistLoginSQLChecker(StaffLoginBox->Text)) {
        if (StaffLoginSQLChecker(StaffLoginBox->Text, StaffOldPasswordBox->Text))
            UpdateStaffPasswordSQL(StaffLoginBox->Text, StaffNewPasswordBox->Text);
        else
            MessageBox::Show("Старый пароль введен неверно");
    }
    else
        MessageBox::Show("Пользователя " + StaffLoginBox->Text + " не существует!");
    EmptyAllTextBox();
}

};
}

```

Модуль «PrintForm.h»

#pragma endregion

```
public: ref class memclass {
    public:
        int ^ID;
        System::String ^name;
        double score;
        System::String ^priority;
        System::String ^form_study;
        System::String ^major;
        System::String ^original;
        System::String ^form_pay;
        System::String ^date;
        memclass() {};
};

System::Collections::Generic::List<memclass^> ^InputLists;

String^ SQLFormatter(){
    //Дефолт выбор
    if ((PayRadioButton1->Checked == true) && (MajorRadioButton1->Checked == true))
        return "SELECT * FROM students WHERE form_pay = 'бюджет' AND
major='ПКС' ORDER BY score DESC;";

    if ((PayRadioButton1->Checked == true) && (MajorRadioButton2->Checked == true))
        return "SELECT * FROM students WHERE form_pay = 'бюджет' AND
major='ИБАС' ORDER BY score DESC;";

    if ((PayRadioButton2->Checked == true) && (MajorRadioButton1->Checked == true))
        return "SELECT * FROM students WHERE form_pay = 'договор' AND
major='ПКС' ORDER BY score DESC;";

    if ((PayRadioButton2->Checked == true) && (MajorRadioButton2->Checked == true))
        return "SELECT * FROM students WHERE form_pay = 'договор' AND
major='ИБАС' ORDER BY score DESC;";

    //Если какой-либо параметр не из формы оплаты
```

```

        if ((PayRadioButton1->Checked == false) && (PayRadioButton2->Checked == false)
        && ((MajorRadioButton1->Checked == true)))

            return "SELECT * FROM students WHERE major='ПКС' ORDER BY score
DESC;";

        if ((PayRadioButton1->Checked == false) && (PayRadioButton2->Checked == false)
        && ((MajorRadioButton2->Checked == true)))

            return "SELECT * FROM students WHERE major='ИБАС' ORDER BY score
DESC;";

        //Если какой-либо параметр не из специальности

        if ((MajorRadioButton1->Checked == false) && (MajorRadioButton2->Checked ==
false) && (PayRadioButton1->Checked == true))

            return "SELECT * FROM students WHERE form_pay = 'бюджет' ORDER BY
score DESC;";

        if ((MajorRadioButton1->Checked == false) && (MajorRadioButton2->Checked ==
false) && (PayRadioButton2->Checked == true))

            return "SELECT * FROM students WHERE form_pay = 'договор' ORDER BY
score DESC;";

        return "SELECT * FROM students ORDER BY score DESC;";

    }

private: void SQLGetter(String^ SQLCommand) {

    SQLiteConnection ^db = gcnew SQLiteConnection();
    db->ConnectionString = GlobalClass::SQLGlobalPatch;

    db->Open();

    SQLiteCommand ^cmdSelect = db->CreateCommand();
    cmdSelect->CommandText = SQLCommand;
    SQLiteDataReader ^data = cmdSelect->ExecuteReader();

    while (data->Read()) {

        memclass ^InputList = gcnew memclass();
        InputList->ID = Int32::Parse(data["ID"]->ToString());
        InputList->name = data["name"]->ToString();
        InputList->score = Double::Parse(data["score"]->ToString());
        InputList->priority = data["priority"]->ToString();
    }
}

```

```

        InputList->form_study = data["form_study"]->ToString();
        InputList->major = data["major"]->ToString();
        InputList->original = data["original"]->ToString();
        InputList->form_pay = data["form_pay"]->ToString();
        InputList->date = data["date"]->ToString();
        InputLists->Add(InputList);
    }
    db->Close();
}

private: System::Void ExitButton_Click(System::Object^ sender, System::EventArgs^ e) {
    this->Hide();
}

private: System::Void ReportButton_Click(System::Object^ sender, System::EventArgs^ e) {

```

```

    SQLGetter(SQLFormatter());

```

```

Object ^ ФорматСтроки = Microsoft::Office::Interop::Word::WdUnits::wdLine;

```

```

auto t = Type::Missing;

```

```

auto WORD = gcnew Microsoft::Office::Interop::Word::ApplicationClass();

```

```

WORD->Visible = true;

```

```

auto Документ = WORD->Documents->Add(t, t, t);

```

```

std::map <int, std::string> TableHeaders;

```

```

TableHeaders[0] = "№";

```

```

TableHeaders[1] = "ФИО";

```

```

TableHeaders[2] = "Средний балл";

```

```

TableHeaders[3] = "Приоритет";

```

```

TableHeaders[4] = "Форма обучения";

```

```

TableHeaders[5] = "Специальность";

```

```

TableHeaders[6] = "Тип аттестата";

```

```

TableHeaders[7] = "Форма оплаты";

```

```

TableHeaders[8] = "Дата";

```

```

//Начинаем работу с Word

```

```

WORD->Selection->ParagraphFormat->Alignment =
Microsoft::Office::Interop::Word::WdParagraphAlignment::wdAlignParagraphCenter;

WORD->Selection->Font->Name = ("Times New Roman");

WORD->Selection->Font->Bold = 1;

WORD->Selection->Font->Size = 20;

WORD->Selection->TypeText("Рейтинг абитуриентов");

WORD->Selection->Font->Size = 12;

WORD->Selection->TypeParagraph();

WORD->Selection->ParagraphFormat->Alignment =
Microsoft::Office::Interop::Word::WdParagraphAlignment::wdAlignParagraphJustify;

WORD->Selection->Font->Bold = false;

WORD->Selection->Font->Size = 9;


int linecounter = 0;

for each (memclass ^InputList in InputLists)

    linecounter++;


//Создаём таблицу

Object ^ ПоказыватьГраницы =
Microsoft::Office::Interop::Word::WdDefaultTableBehavior::wdWord9TableBehavior;

Object ^ РегулирШирины =
Microsoft::Office::Interop::Word::WdAutoFitBehavior::wdAutoFitWindow;

Microsoft::Office::Interop::Word::Range ^ wrdRng = WORD->Selection->Range;

WORD->ActiveDocument->Tables->Add(wrdRng, linecounter+1, 9, ПоказыватьГраницы,
РегулирШирины);


//Заполнение заголовков таблицы

for (int i = 0; i < 9; i++) {

    String^ buf = gcnew System::String(TableHeaders[i].c_str());

    WORD->ActiveDocument->Tables[1]->Cell(1, i + 1)->Range->InsertAfter(buf);

}

//Заполнение ячеек таблицы

int i = 2;

for each (memclass ^InputList in InputLists) {

    WORD->ActiveDocument->Tables[1]->Cell(i, 1)->Range-
>InsertAfter(System::Convert::ToString(i-1));

```

```

WORD->ActiveDocument->Tables[1]->Cell(i, 2)->Range->InsertAfter(InputList-
>name);

WORD->ActiveDocument->Tables[1]->Cell(i, 3)->Range-
>InsertAfter(System::Convert::ToString(InputList->score));

WORD->ActiveDocument->Tables[1]->Cell(i, 4)->Range->InsertAfter(InputList-
>priority);

WORD->ActiveDocument->Tables[1]->Cell(i, 5)->Range->InsertAfter(InputList-
>form_study);

WORD->ActiveDocument->Tables[1]->Cell(i, 6)->Range->InsertAfter(InputList-
>major);

WORD->ActiveDocument->Tables[1]->Cell(i, 7)->Range->InsertAfter(InputList-
>original);

WORD->ActiveDocument->Tables[1]->Cell(i, 8)->Range->InsertAfter(InputList-
>form_pay);

WORD->ActiveDocument->Tables[1]->Cell(i, 9)->Range->InsertAfter(InputList-
>date);

i++;
    }
}

```

```

private: System::Void PrintForm_Load(System::Object^ sender, System::EventArgs^ e) {
    InputLists = gcnew System::Collections::Generic::List<memclass^>();
    PayRadioButton1->Checked = false;
    MajorRadioButton1->Checked = false;
}

};
}

```

Модуль «RatingForm.h»

#pragma endregion

```

private: DataTable^ FillTableRatingForm() {
    std::map <int, std::string> rows_formatter;

    rows_formatter[0] = "№ заявления";
    rows_formatter[1] = "ФИО";
    rows_formatter[2] = "Сред. балл";
    rows_formatter[3] = "Приоритет";
    rows_formatter[4] = "Форма обучения";
}

```

```

rows_formatter[5] = "Специальность";
rows_formatter[6] = "Тип аттестата";
rows_formatter[7] = "Форма оплаты";
rows_formatter[8] = "Дата ";

```

```

SQLiteConnection ^db = gcnew SQLiteConnection();

```

```

    db->ConnectionString = GlobalClass::SQLGlobalPatch;

```

```

    db->Open();

```

```

    SQLiteCommand ^cmdSelect = db->CreateCommand();

```

```

    cmdSelect->CommandText = "SELECT * FROM students;";

```

```

    SQLiteDataReader ^reader = cmdSelect->ExecuteReader();

```

```

    DataTable ^table;

```

```

    DataColumn ^column;

```

```

    DataRow ^row;

```

```

    table = gcnew DataTable();

```

```

    vector<String^>^ nameColumns = gcnew vector<String^>();

```

```

    for (int i = 0; i < reader->FieldCount; i++) {

```

```

        String^ buf_row = gcnew System::String(rows_formatter[i].c_str());

```

```

        nameColumns->push_back(buf_row);

```

```

        column = gcnew DataColumn(nameColumns->at(i), String::typeid);

```

```

        table->Columns->Add(column);

```

```

    }

```

```

    while (reader->Read()) {

```

```

        row = table->NewRow();

```

```

        for (int i = 0; i < reader->FieldCount; i++) {

```

```

            row[nameColumns->at(i)] = reader->GetValue(i)->ToString();

```

```

            reader->GetValue(i)->ToString();

```

```

        }

```

```

        table->Rows->Add(row);

```

```

    }

```

```

    db->Close();

```

```

    return table;

```

```

}

```

```

private: System::Void ExitButton_Click(System::Object^ sender, System::EventArgs^ e) {

```

```

    this->Hide();

```

```

    }

    private: System::Void RatingForm_Load(System::Object^ sender, System::EventArgs^ e) {
        dataGridView1->DataSource = FillTableRatingForm();
    }

    private: System::Void dataGridView1_CellContentClick(System::Object^ sender,
System::Windows::Forms::DataGridViewCellEventArgs^ e) {
    }

    private: System::Void PrinterButton_Click(System::Object^ sender, System::EventArgs^ e) {

        int index = dataGridView1->CurrentCell->RowIndex;

        String^ unic_student_id = dataGridView1->Rows[index]->Cells[0]->Value->ToString();

        String^ unic_student_fio = dataGridView1->Rows[index]->Cells[1]->Value->ToString();
        DEMKA::ChoicePrintTypeForm^ChoicePrintTypeForm_obj = gcnew
DEMKA::ChoicePrintTypeForm();

        ChoicePrintTypeForm_obj->PublicStudentID = unic_student_id;
        ChoicePrintTypeForm_obj->PublicStudentFIO = unic_student_fio;
        ChoicePrintTypeForm_obj->ShowDialog();

        this->Show();
    }
};
}

```

Модуль «RemoveStaffForm.h»

#pragma endregion

```

private: void EmptyAllTextBox() {
    StaffLoginBox->Text = "";
    StaffPasswordBox->Text = "";
    MasterPasswordBox->Text = "";
}

private: void RemoveStaffSQL(String^ login_str, String^ password_str) {

    SQLiteConnection ^db = gcnew SQLiteConnection();

```



```

String^ MD5login_str = GlobalClass::getMD5String(GlobalClass::toBase64(login_str));

String^ MD5password_str = GlobalClass::getMD5String(GlobalClass::toBase64(password_str));

String^ SQL_STRING = "DELETE FROM staff WHERE login = " + MD5login_str + " AND
password = " + MD5password_str + """;

db->ConnectionString = GlobalClass::SQLGlobalPatch;

db->Open();

SQLiteCommand ^cmdInsertValue = db->CreateCommand();
cmdInsertValue->CommandText = SQL_STRING;

cmdInsertValue->ExecuteNonQuery();

db->Close();

MessageBox::Show("Успешное удаление пользователя АИС");

this->Hide();

}

private: bool StaffPasswordSQLChecker(String^ password_str) {

    SQLiteConnection ^db = gcnew SQLiteConnection();

    String^ MD5password_str = GlobalClass::getMD5String(GlobalClass::toBase64(password_str));

    String^ BufValidationStr = "DEMKA";
    String^ SQL_STRING = "SELECT * FROM staff WHERE password = " + MD5password_str +
    """;

    db->ConnectionString = GlobalClass::SQLGlobalPatch;

    db->Open();

    SQLiteCommand ^cmdSelect = db->CreateCommand();

    cmdSelect->CommandText = SQL_STRING;

    SQLiteDataReader ^data = cmdSelect->ExecuteReader();

    while (data->Read())

        BufValidationStr = data->GetValue(2)->ToString();

    db->Close();

    if (BufValidationStr != "DEMKA")

        return true;

    return false;

}

private: bool StaffLoginSQLChecker(String^ login_str) {

    SQLiteConnection ^db = gcnew SQLiteConnection();

    String^ MD5login_str = GlobalClass::getMD5String(GlobalClass::toBase64(login_str));

```

```

String^ BufValidationStr = "DEMKA";
String^ SQL_STRING = "SELECT * FROM staff WHERE login ='" + MD5login_str + "'";

db->ConnectionString = GlobalClass::SQLGlobalPatch;

db->Open();

SQLiteCommand ^cmdSelect = db->CreateCommand();

cmdSelect->CommandText = SQL_STRING;

SQLiteDataReader ^data = cmdSelect->ExecuteReader();

while (data->Read())
    BufValidationStr = data->GetValue(1)->ToString();

db->Close();

if (BufValidationStr != "DEMKA")
    return true;

return false;
}

private: bool MasterPasswordChecker(String^ master_str) {
    return
(GlobalClass::getMD5String(GlobalClass::toBase64(master_str))==GlobalClass::MasterGlobalPassword) ? true :
false;
}

private: System::Void ExitButton_Click(System::Object^ sender, System::EventArgs^ e) {
    this->Hide();
}

private: System::Void StaffRemoveButton_Click(System::Object^ sender, System::EventArgs^ e) {
    if ((StaffLoginSQLChecker(StaffLoginBox->Text)) &&
(StaffPasswordSQLChecker(StaffPasswordBox->Text)) && (MasterPasswordChecker(MasterPasswordBox-
>Text)))

        RemoveStaffSQL(StaffLoginBox->Text, StaffPasswordBox->Text);

    else if (StaffLoginSQLChecker(StaffLoginBox->Text) == false)

        MessageBox::Show("Введенного пользователя нет в системе!");

    else if (MasterPasswordChecker(MasterPasswordBox->Text) == false)

        MessageBox::Show("Мастер-пароль введен неверно!");

    else

        MessageBox::Show("Неверный пароль пользователя АИС!");

    EmptyAllTextBox();
}

```

```

    }

private: System::Void RemoveStaffForm_Load(System::Object^ sender, System::EventArgs^ e) {

    MessageBox::Show("Внимание!\nДля удаления пользователя из АИС требуется:\n-Имя
пользователя\n-Пароль пользователя\n-Мастер-пароль\nЕсли один из этих параметров недоступен, то
УДАЛЕНИЕ ПОЛЬЗОВАТЕЛЯ НЕВОЗМОЖНО");

}

};

}

Модуль «StaffLoginForm.h»

#pragma endregion

private: void EmptyAllTextBox() {

    LoginBox->Text = "";

    PasswordBox->Text = "";

}

private: bool StaffSQLChecker(String^ login_str, String^ password_str){

    SQLiteConnection ^db = gcnew SQLiteConnection();

    String^ BufChecker = "DEMKA";

    String^ MD5Base64login_str = GlobalClass::getMD5String(GlobalClass::toBase64(login_str));

    String^ MD5Base64password_str =
GlobalClass::getMD5String(GlobalClass::toBase64(password_str));

    String^ SQL_STRING = "SELECT * FROM staff WHERE login = '"+ MD5Base64login_str +'"
AND password = '"+ MD5Base64password_str +'"";
    db->ConnectionString = GlobalClass::SQLGlobalPatch;

    db->Open();

    SQLiteCommand ^cmdSelect = db->CreateCommand();

    cmdSelect->CommandText = SQL_STRING;

    SQLiteDataReader ^data = cmdSelect->ExecuteReader();

    while (data->Read())

        BufChecker = data->GetValue(0)->ToString();

    db->Close();

    if (BufChecker != "DEMKA")

        return true;

    return false;
}

```

```

    }

private: System::Void LoginButton_Click(System::Object^ sender, System::EventArgs^ e) {
    if ((StaffSQLChecker(LoginBox->Text, PasswordBox->Text)) == true) {
        MessageBox::Show("Успешная авторизация");
        MainForm^MainForm_obj = gcnew MainForm();

        MainForm_obj->ShowDialog();

        this->Close();
    }
    else{
        MessageBox::Show("Неверный пароль!");

        EmptyAllTextBox();
    }
}

private: System::Void StafLoginForm_Load(System::Object^ sender, System::EventArgs^ e){
    GlobalClass::SQLGlobalPatch = "Data Source=../database_vs.db";
    GlobalClass::MasterGlobalPassword = "9A-F0-3C-0D-AD-FF-B9-87-82-03-3B-E7-14-CA-95-
9B";
    SQLiteConnection ^db = gcnew SQLiteConnection();

    try
    {
        db->ConnectionString = GlobalClass::SQLGlobalPatch;

        db->Open();

        SQLiteCommand ^cmdInsertValue = db->CreateCommand();

        cmdInsertValue->CommandText = "CREATE TABLE IF NOT EXISTS staff" +
            "(ID INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, login
            TEXT, password TEXT);" +
            "INSERT INTO staff(login,password) SELECT 'DB-69-FC-03-9D-CB-D2-96-
            2C-B4-D2-8F-58-91-AA-E1', 'DB-69-FC-03-9D-CB-D2-96-2C-B4-D2-8F-58-91-AA-E1'" +
            "WHERE NOT EXISTS(SELECT 1 FROM staff WHERE login = 'DB-69-FC-
            03-9D-CB-D2-96-2C-B4-D2-8F-58-91-AA-E1');"

        cmdInsertValue->ExecuteNonQuery();

        db->Close();
    }
    finally

```

```
        {
            delete (IDisposable^)db;
        }
    }
};
}
```

ПРИЛОЖЕНИЕ В

ПРЕЗЕНТАЦИЯ К ЗАЩИТЕ КУРСОВОГО ПРОЕКТА

На рисунках В.1-В.11 показаны содержимое слайдов презентации к защите курсового проекта.

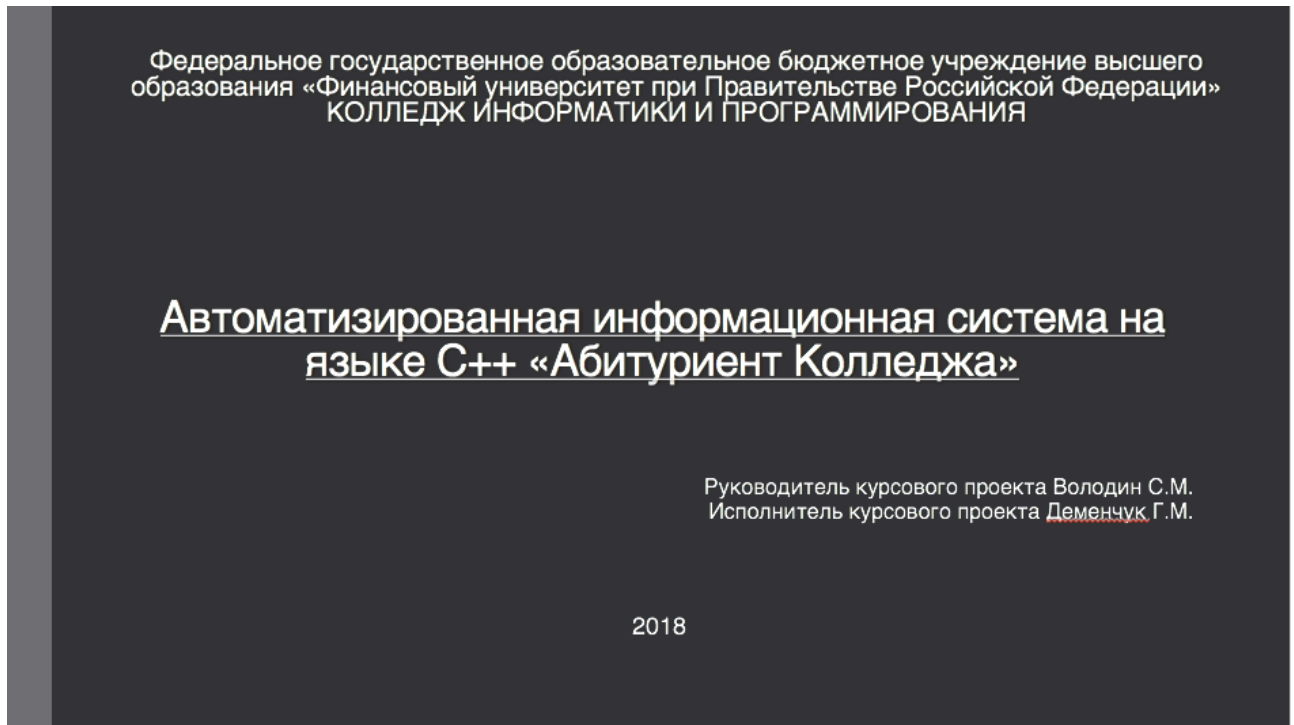


Рис. В.1 – Титульный лист

Содержание

- Цели курсового проекта
- Задачи курсового проекта
- Входные и выходные данные
- Информационная модель программы
- Логическая модель задачи
- Логическая модель программы
- Интерфейс программы
- Достоинства проекта
- Результаты выполнения проекта



Рис. В.2 – Содержание

Задачи курсового проекта

1. Информатизация регистрации данных об абитуриентах:
Предполагает ведение базы данных с несколькими таблицами, содержащими информацию об абитуриентах колледжа. На основании полученных данных ведется формирование рейтинга абитуриентов с определенной выборкой по направлению (ПКС/ИБАС) или по типу аттестата (оригинал/копия)
2. Редактирование базы данных — удаление, обновление, добавление данных абитуриентов.
3. Формирование отчетов. Создание отчетов по каждому абитуриенту при приеме документов в колледж, а также при формировании рейтинга абитуриентов с их последующим выводом на печать.



Рис. В.3 – Задачи курсового проекта

Входные и выходные данные

- Входные данные — содержание элементов Windows Forms, как: TextBox, Combobox и RadioButton.
- Выходные данные — база данных SQLite3, документы Microsoft Word, а также содержание элементов WindowsForms GridTable.

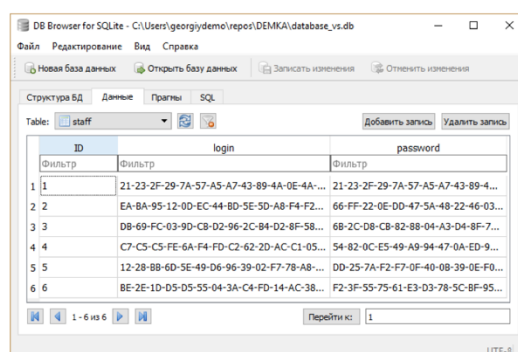


Рис. В.4 – Входные и выходные данные

Информационная модель программы

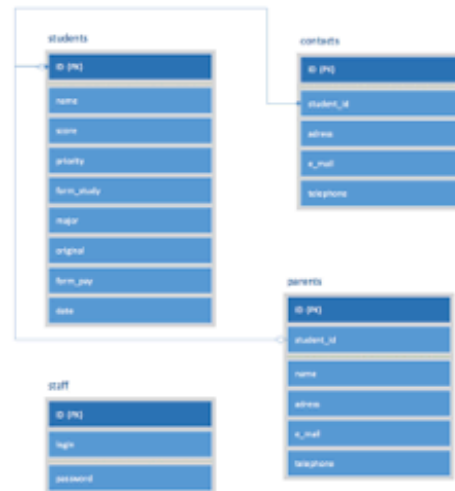


Рис. В.5 – Информационная модель программы

Логическая модель задачи

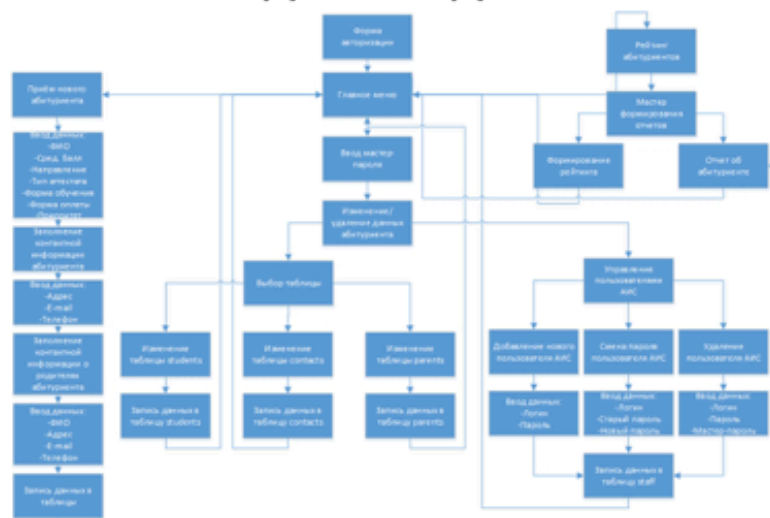


Рис. В.6 – Логическая модель задачи

Логическая модель программы

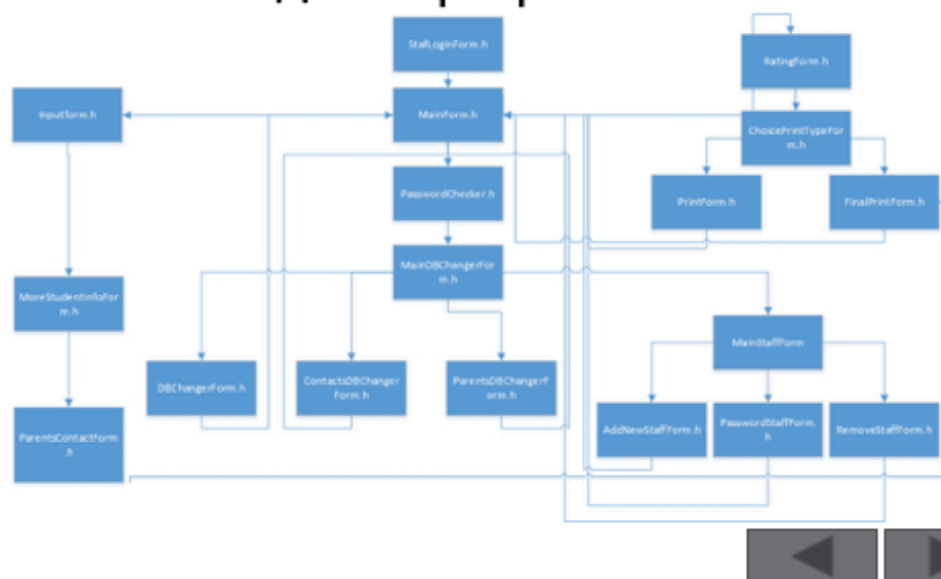


Рис. В.7 – Логическая модель программы

Интерфейс программы

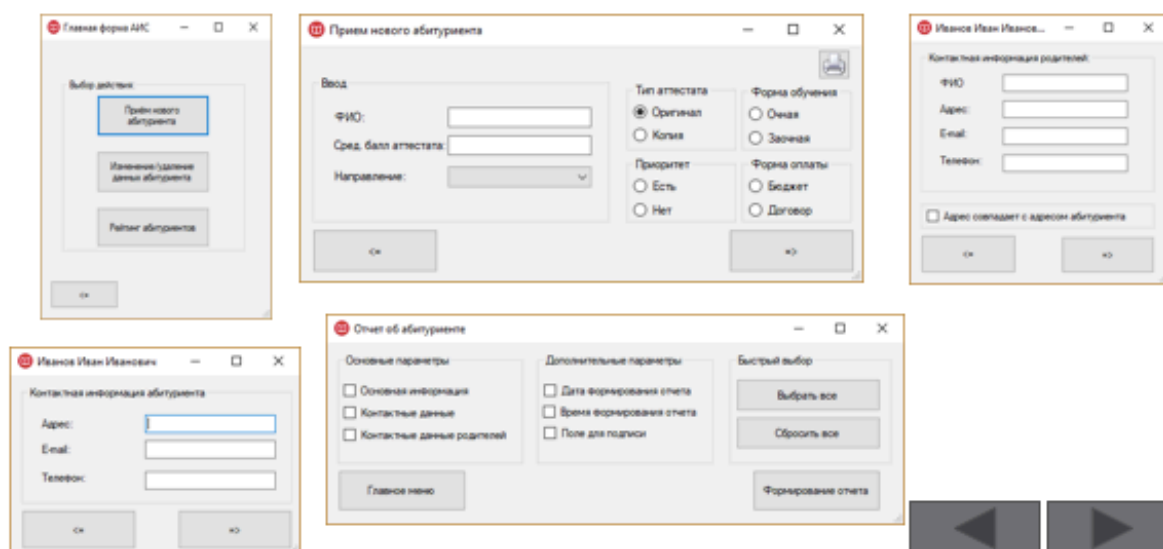


Рис. В.8 – Интерфейс программы

Достоинства проекта

1. За счет C++/CLI с Windows Forms стало возможным реализовать интуитивно понятный интерфейс для взаимодействия с пользователем. Для использования программы пользователю необходимы лишь основы работы с компьютером.
2. Динамическое формирование отчетов в Microsoft Office Word: пользователю предоставляется возможность выбора пунктов для вывода.
3. Использование компактной встраиваемой реляционной база данных SQLite3, которая идеально подходит для портируемых и не предназначенных для масштабирования приложений.



Рис. В.9 – Достоинства проекта

Результаты выполнения проекта

В ходе разработки программы были получены и усовершенствованы навыки работы со следующими составляющими:

- ЯП C++/CLI
- СУБД SQLite3
- ООП

Цели и задачи, поставленные при выполнении курсового проекта, выполнены с соблюдением всех предъявленных требований в установленные сроки.



Рис. В.10 – Результаты выполнения проекта

Спасибо за внимание

Рис. В.11 – Последний слайд