|  |  |  |
| --- | --- | --- |
|  |  |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |

Институт Информационных технологий

Кафедра Математического обеспечения и стандартизации информационных технологий

**КУРСОВАЯ РАБОТА**

по дисциплине «Объектно-ориентрованное программирование»

**Тема курсовой работы** Проектирование и реализация программной системы с использованием объектного подхода

**Студент группы** ИКБО-12-17 Лисовой Андрей Анатольевич

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

(подпись студента)

**Руководитель курсовой работы** Преподаватель Миронов А.Н.

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

(подпись руководителя)

Работа представлена к защите «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_ 2018 г.

Допущен к защите «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2018 г.

Москва 2018

|  |
| --- |
| C:\Documents and Settings\admin\Рабочий стол\МИРЭА_ЭМБЛЕМА_приказ.JPG |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«Московский технологический университет»** МИРЭА |

Институт Информационных технологий

Кафедра Математического обеспечения и стандартизации информационных технологий

Утверждаю

Заведующий кафедрой МОСИТ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Головин С.А.

«\_\_\_» \_\_\_\_\_\_\_\_\_ 2018 г.

**ЗАДАНИЕ**

**на выполнение курсовой работы по дисциплине**

«Объектно-ориентрованное программирование»

Студент Лисовой Андрей Анатольевич Группа ИКБО-12-17

**Тема работы:** Проектирование и реализация программной системы с использованием объектного подхода

**Исходные данные:** задание на курсовую работу,тема №1, вариант №1.12. Разработать систему Контроля присутствия сотрудника на рабочем месте. Функционал системы: Регистрация сотрудников компании в системе; Формирования графика работы сотрудников; Регистрация прохода сотрудника через электронную систему (время прихода, время выхода, дата); Сформировать список сотрудников компании, которые в течение месяца нарушили график   
работы, указав даты и часы.

**Перечень вопросов, подлежащих разработке, и обязательного графического материала:**

Моделирование объектной системы с использованием обозначений языка моделирования UML

Реализация программной системы средствами языка С++

Отчет по курсовой работе в виде расчетно-пояснительной записки .

**Срок представления к защите курсовой работы:** до «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2018 г.

**Задание на курсовую работу выдал** \_\_\_\_\_\_\_\_\_\_\_\_ (Миронов А.Н.)

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2018 г.

**Задание на курсовую работу получил**  \_\_\_\_\_\_\_\_\_\_\_\_ (Лисовой А.А.)

**Оглавление**

[РЕФЕРАТ 4](#_Toc515969000)

[ВВЕДЕНИЕ 5](#_Toc515969001)

[1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 7](#_Toc515969002)

[1.1 ГЛАВНЫЕ ОСОБЕННОСТИ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПОДХОДА 7](#_Toc515969003)

[1.2 ОСНОВНЫЕ ПРИНЦИПЫ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПОДХОДА 7](#_Toc515969004)

[1.3 ПОНЯТИЕ КЛАССОВ И ОБЪЕКТОВ 9](#_Toc515969005)

[1.4 ПРИНЦИПЫ ОБЪЕКТНОЙ ДЕКОМПОЗИЦИИ 9](#_Toc515969006)

[1.5 ОСНОВНЫЕ ХАРАКТЕРИСТИКИ ОБЪЕКТНОЙ МОДЕЛИ 11](#_Toc515969007)

[2. ПРОЕКТНАЯ ЧАСТЬ 13](#_Toc515969008)

[2.1 ПОСТАНОВКА ЗАДАЧИ 13](#_Toc515969009)

[2.2 ПРОЕКТНОЕ РЕШЕНИЕ 13](#_Toc515969010)

[2.3 ДИАГРАММЫ КЛАССОВ 17](#_Toc515969011)

[2.4 ОПИСАНИЕ СТРУКТУРЫ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА 19](#_Toc515969012)

[3. ЭКСПЕРИМЕНТАЛЬНАЯ ЧАСТЬ 22](#_Toc515969013)

[3.1 ОПИСАНИЕ ПОДХОДА К ТЕСТИРОВАНИЮ 22](#_Toc515969014)

[3.2 ТЕСТИРОВАНИЕ ПРОГРАММЫ 22](#_Toc515969015)

[3.3 ИНСТРУКЦИЯ ПРОГРАММИСТУ 27](#_Toc515969016)

[ЗАКЛЮЧЕНИЕ 28](#_Toc515969017)

[СПИСОК ЛИТЕРАТУРЫ 29](#_Toc515969018)

[ПРИЛОЖЕНИЯ 30](#_Toc515969019)

[Приложение 1. Модуль класса Stuff\_Worker\_Time. 30](#_Toc515969020)

[Приложение 2. Модуль класса Stuff\_Member. 31](#_Toc515969021)

[Приложение 3. Модуль подпрограмм Sub\_Programs: 34](#_Toc515969022)

[Приложение 4. Главная форма Interface\_Main: 42](#_Toc515969023)

[Приложение 4. Форма добавления сотрудника Interface\_Add: 53](#_Toc515969024)

[Приложение 6. Форма регистрации прохода Add\_Appearance: 55](#_Toc515969025)

[Приложение 7. Форма управления нарушениями Violation\_Manager: 59](#_Toc515969026)

# РЕФЕРАТ

Объем сей курсовой работы занимает порядка 63 страниц. Эта курсовая работа имеет три основных раздела – теоретический, практический и экспериментальный. Так же она включает в себя графические материалы: 2 таблицы и 19 рисунков.

Теоретическая часть включает в себя сокращенные сведения о концепции объектно-ориентированного программирования. Описаны основные принципы этой парадигмы, а также её основные свойства. Были даны определения классов и объектов, а также самой объектной декомпозиции технического задания. Были подробно рассмотрены основные свойства классов и объектов.

Проектная часть служит для постановки задачи и проведения объектной декомпозиции. В ходе решения задачи были разработаны соответствующие ей классы, информация о каждом из них приведена в соответствующих таблицах. Приведены диаграммы классов и диаграмма иерархии их наследования. Разработан и описан пользовательский графический интерфейс программы, демонстрирующий работу созданных классов.

Экспериментальная часть работы описывает подходы тестирования проекта. Тестирование происходит путем выполнения различных соответствующих задаче операции через взаимодействие с разработанным графическим пользовательским интерфейсом. Результаты тестирования были приведены с помощью скриншотов, подтверждающих корректную работу программы.

В приложении к курсовой работе приведен исходный код разработанной программы в модульном виде.

# ВВЕДЕНИЕ

Для разработки различного программного обеспечения на сей день существуют несколько основных парадигм программирования. Среди них принято выделять следующие:

1. Императивная парадигма – это модель, в которой программа являет собой набор четких последовательных команд для выполнения компьютером, причем данные, необходимые для её работы, могут записываться или сохраняться в память.
2. Декларативная парадигма – это модель, в которой в первую очередь задается специфика поставленной для решения задачи, а также ожидаемый результат. Языки этой парадигмы не содержат переменных и оператора присваивания.
3. Структурная парадигма – это модель, в которой главным является представление любой программы, как совокупность исполняемых блоков, организованных в единую иерархическую структуру. Строго запрещен оператор безусловного перехода.
4. Функциональная парадигма – это модель, в которой любое вычисление определяется как процесс вычисления значения математических функций. Эта парадигма не предполагает явного хранения состояния программы.
5. Логическая парадигма – эта модель основана на автоматическом доказательстве теорем, а также на аппарате математической логики с использованием его принципов.
6. Объектно-Ориентированная парадигма – эта модель основана на представлении любой программы как совокупности объектов, где каждый объект суть экземпляр класса, а классы создают иерархию наследования.

Каждая из представленных парадигм была создана для решения конкретного ряда задач, посему было бы очень сложно выделить среди них наиболее лучшую. Но, именно объектно-ориентированная парадигма, как показывает практика, является наиболее многофункциональной и универсальной из всех ныне существующих.

Объектно-Ориентированное Программирование (ООП) можно рассматривать в качестве закономерного итога длительной эволюции более ранних парадигм программирования, вызванного постоянным усложнением ставящихся перед программистами задач и требующего все более надежных и универсальных решений.

Целью сей курсовой работы является разработка приложения с пользовательским графическим интерфейсом, которое призвано продемонстрировать работоспособность разработанных классов, а также их организацию в единую программную систему учета и контроля посещаемости сотрудников. Работа выполнена в среде CLR для языка C++ в интегрированной среде разработки Microsoft Visual Studio 2017.

# ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

## ГЛАВНЫЕ ОСОБЕННОСТИ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПОДХОДА

Изначально объектно-ориентированная парадигма была разработана с целью избавиться от недостатков другой, на тот момент доминирующей в мире программирования парадигмы – процедурной. В отличие от неё, объектно-ориентированный стиль подразумевает написание не вызывающих друг друга блоков(процедур), а программных классов, наделенных необходимым для решения конкретной задачи поведением. Создавая экземпляры этих классов, программист реализует выполнение ими определенных задач, а также принятие, обработку и передачу ими данных.

Поистине, сложной стала бы разработка сложных систем, если бы не существовала такая дисциплина, как объектно-ориентированное проектирование. Она предназначена для максимально эффективной и корректной разработки сложных программных систем, подразумевая ещё на этапе проектирования сложной системы её анализ с точки зрения объектно-ориентированного подхода.

## ОСНОВНЫЕ ПРИНЦИПЫ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПОДХОДА

Объектно-ориентированная парадигма базируется на четырёх основных принципах, каждый из которых следует из предыдущего:

1. Инкапсуляция – это сокрытие реализации уже написанных блоков кода для большей безопасности и удобства работы. Она подразумевает тот факт, что для использования уже написанного класса абсолютно не обязательно видеть его внутреннюю структуру, достаточно иметь доступ к методам интерфейса данного класса. Именно инкапсуляция позволяет программисту решить, какие свойства, методы и классы будут открыты классу-клиенту, а какие должны быть скрыты.
2. Наследование – это возможность создавать объекты класса на основе более старого уже существующего класса. Классу-наследнику будут доступны все методы класса-родителя, включая обращение к полям. Это позволяет, изменив функциональность класса-родителя также изменить и функциональность всех его классов-наследников. Наследование позволяет существенно сократить объем программирования вручную.
3. Полиморфизм – это механизм, делающий возможным существование одноименных методов и полей при описании нескольких различных классов. Так же это возможность объектов с одинаковой спецификацией иметь различную реализацию. Полиморфизм существует для повышения возможности повторного использования уже написанного кода.
4. Абстракция - В ООП это придание объекту класса характеристик, которые резко выделяют его на фоне остальных объектов, определяя его концептуальные границы. Абстрагирование - В ООП это способ выделить набор приоритетных характеристик объекта, исключая из рассмотрения менее значимые. Следовательно, абстракция — это набор таких характеристик.

## ПОНЯТИЕ КЛАССОВ И ОБЪЕКТОВ

Класс – это ключевое понятие в объектно-ориентированном программировании. Это основополагающий элемент, в рамках которого осуществляется проектирование программ. Класс содержит в себе различные данные и код, управляющий этими данными. Почти всегда класс служит для описания объекта или явления реального мира. Как и реальный объект, класс содержит свой набор параметров и характеристик. Каждый такой параметр называется полем класса. Также класс способен проводить различные действия со своими полями с помощью методов класса.

Конструктор класса - это метод, вызываемый при создании нового объекта данного класса и используемый для заполнения полей класса значениями, а также для начальных операций, если они необходимы. После создания объекта вызов конструктора становится невозможным. Помимо этого, сей метод никогда не возвращает никакого значения.

Конструктор по умолчанию – это пустой конструктор без параметров в вызове. Он всегда присутствует в любом классе, даже если он не был объявлен явно. Конструктор по умолчанию вызывается автоматически, когда отсутствуют другие конструкторы.

Деструктор – это специальный метод класса, служащий для освобождения памяти объектом при его удалении.

Объект – это экземпляр одного или нескольких классов. Он имеет определённое состояние и поведение, а также заданные значения свойств и операций над ними.

## ПРИНЦИПЫ ОБЪЕКТНОЙ ДЕКОМПОЗИЦИИ

Объектная декомпозиция – это процесс представления предметной области задания в виде совокупности объектов, производящих обмен данными между собой. При объектной декомпозиции между объектами устанавливается следующие типы отношений:

1. Использования – первый объект (активный) передает сообщение другому (пассивному), между ними могут быть объекты-посредники.
2. Включения – если объект является результатов декомпозиции более сложного объекта.

Объектная декомпозиция может применяться многократно, или быть многоуровневой. Это означает, что каждый объект может рассматриваться как система, которая состоит из элементов, взаимодействующих друг с другом через передачу различных данных. При многоуровневой декомпозиции для каждого уровня мы имеем объекты с более простым поведением, что позволяет разрабатывать системы повышенной сложности дифференцированно. Декомпозиция применяется в тех случаях, когда рассматривается существующая система или проектируется новая. При этом применяются два фундаментальных понятия: анализ и синтез систем. Задачи анализа определяются как изучение свойств и поведения системы в зависимости от ее структуры и значений параметров, исходя из заданных свойств системы. Задачи синтеза сводятся к выбору структуры и значений параметров, исходя из заданных свойств системы.

## ОСНОВНЫЕ ХАРАКТЕРИСТИКИ ОБЪЕКТНОЙ МОДЕЛИ

Основными принципами построения объектной модели это: абстрагирование, инкапсуляция, модульность, иерархичность, типизация, параллелизм и сохраняемость. Каждый из этих принципов сам по себе не является новым, но в объектной модели они впервые применены в качестве единой системы.

1. Абстрагирование - выделяет существенные характеристики некоторого объекта, отличающие его от всех других видов объектов и, таким образом, четко определяет его функциональные границы с точки зрения наблюдателя.
2. Инкапсуляция - это процесс отделения друг от друга элементов объекта, определяющих его устройство и поведение. Она служит для того, чтобы изолировать обязательства абстракции от их реализации.
3. Модульность - это свойство системы, подразумевающее её дифференцирование на внутренне связные, но при этом слабо связанные между собой модули. Структура модуля должна быть достаточно простой для восприятия, а реализация каждого модуля не должна зависеть от реализации других модулей. Так же должны быть приняты меры для облегчения процесса внесения изменений там, где они наиболее вероятны в будущем.
4. Иерархия - это процесс упорядочивания абстракций, путём расположения их по уровням. Основными видами иерархических структур применительно к сложным системам являются структура классов и структура объектов.
5. Типизация - это способ защититься от использования объектов одного класса вместо другого, или по крайней мере взять управление таким использованием в свои руки. Тип - точная характеристика свойств, включая структуру и поведение, относящаяся к некоторой совокупности объектов. Тип и класс – не являются тождественными понятиями. Можно сказать, что класс реализует понятие типа.
6. Параллелизм - это свойство, отличающее активные объекты от пассивных. Параллелизм позволяет использовать несколько компьютеров для решения задачи или задействовать многозадачность на многопроцессорном компьютере.
7. Сохраняемость - способность объекта существовать во времени, переживая породивший его процесс, и (или) в пространстве, перемещаясь из своего первоначального адресного пространства в другое.

# ПРОЕКТНАЯ ЧАСТЬ

## 2.1 ПОСТАНОВКА ЗАДАЧИ

Задание:

1. Выделить и разработать необходимые для решения данной конкретной задачи классы.
2. Организовать работу с данными системы, включая их обработку, систематизацию, добавление/удаление, сохранение и чтение данных системы в файл/из файла.
3. Разработать приложение с пользовательским визуальным интерфейсом, позволяющий пользователю максимально эффективно и комфортно взаимодействовать с разработанной системой.

## 2.2 ПРОЕКТНОЕ РЕШЕНИЕ

Для начала необходимо провести объектную декомпозицию поставленной для выполнения задачи. Разработаем класс Stuff\_Worker\_Time (содержит часы и минуты) и класс Stuff\_Member (Содержит данные о сотруднике, а также четыре экземпляра класса Stuff\_Worker\_Time (Время прихода/ухода по графику и реальное время прихода/ухода)).

1. Класс Stuff\_Worker\_Time:

Таблица 1.

|  |  |  |  |
| --- | --- | --- | --- |
| Имя члена класса | Модификатор доступа | Тип данных члена класса/возвращаемого значения | Описание |
| Поля | | | |
| Hour | private | unsigned short int | Часы |
| Minutes | private | unsigned short int | Минуты |
| Методы | | | |
| Stuff\_Worker\_Time() | public | - | Конструктор без параметров |
| Stuff\_Worker\_Time(параметры) | public | - | Конструктор с параметрами |
| set\_Hour | public | void | Установить часы |
| set\_Minutes | public | void | Установить минуты |
| get\_Hour | public | unsigned short int | Получить часы |
| get\_Minutes | public | unsigned short int | Получить минуты |
| ~Stuff\_Worker\_Time | public | - | Деструктор |

1. Класс Stuff\_Member:

Таблица 2.

|  |  |  |  |
| --- | --- | --- | --- |
| Имя члена класса | Модификатор доступа | Тип данных члена класса/возвращаемого значения | Описание |
| Поля | | | |
| Number | private | unsigned int | Номер сотрудника |
| Name[20] | private | char | Имя сотрудника |
| Surname[20] | private | char | Фамилия сотрудника |
| WorkBegin | private | Stuff\_Worker\_Time | Начало работы по графику |
| WorkEnd | private | Stuff\_Worker\_Time | Конец работы по графику |
| Advent | private | Stuff\_Worker\_Time | Реальное время прихода |
| Leave | private | Stuff\_Worker\_Time | Реальное время ухода |
| Attendance[20] | private | char | Отметка о посещаемости за день |
| Методы | | | |
| Stuff\_Member | public | - | Конструктор без параметров |
| Stuff\_Member(параметры) | public | - | Конструктор с параметрами |
| set\_Number | public | void | Установить номер |
| set\_Name | public | void | Установить имя |
| set\_Surname | public | void | Установить фамилию |
| set\_Attendance | public | void | Установить посещаемость |
| set\_WorkBegin | public | void | Установить время начала работы по графику |
| set\_WorkEnd | public | void | Установить время конца работы по графику |
| set\_Advent | public | void | Установить реальное время прихода |
| set\_Leave | public | void | Установить реальное время уход |
| get\_Number | public | unsigned int | Получить номер |
| get\_Name | public | char | Получить имя |
| get\_Surname | public | char | Получить фамилию |
| get\_Attendance | public | char | Получить посещаемость |
| get\_WorkBegin | public | Stuff\_Worker\_Time | Получить время прихода по графику |
| get\_WorkEnd | public | Stuff\_Worker\_Time | Получить время ухода по графику |
| get\_Advent | public | Stuff\_Worker\_Time | Получить реальный приход |
| get\_Leave | public | Stuff\_Worker\_Time | Получить реальный уход |
| ~Stuff\_Member | public | - | Деструктор |

Класс Stuff\_Member является наследником класса Stuff\_Worker\_Time и, следовательно, наследует все члены класса-родителя. Итого, в этот класс входят такие поля, как: Индивидуальный номер, Имя, Фамилия, четыре экземпляра класса Stuff\_Worker\_Time, ответственных за время прихода и ухода сотрудника с работы, а также отметку о посещении.

Для того, чтобы организовать наши данные в систему журнала посещаемости, используем встроенный шаблонный класс из пространства имен std - vector. Пусть все данные нашего электронного журнала (все дни в течение года) будут представлены, как вектор векторов, где главный вектор будет представлять из себя динамический массив векторов, с длиной, равной количеству дней в текущем году, где каждый вложенный вектор (один рабочий день) будет содержать в себе динамический массив экземпляров класса Stuff\_Member с данными о посещаемости каждого объекта. Итого, мы получаем таблицу, где можно наглядно увидеть данные о посещаемости всех сотрудников по дням в течение этого года. Так же, такая организация хранения данных во время работы программы значительно облегчает такие операции над данными, как удаление или добавление нового сотрудника в систему, а также запись и чтение данных из файла по завершению/началу работы программы.

## ДИАГРАММЫ КЛАССОВ

Данные UML-диаграммы классов были созданы с помощью ресурса Draw.io:

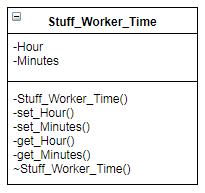


Рис. 1 – Диаграмма класса Stuff\_Worker\_Time

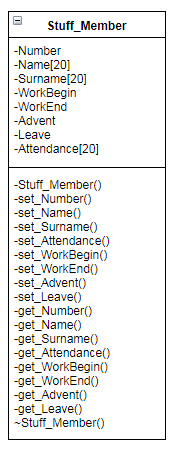


Рис. 2 – Диаграмма класса Stuff\_Member

Так как класс Object(System)является исходным базовым классом для всех классов платформы .NET Framework и корнем иерархии типов, справедливым будет утверждение, что все созданные пользовательские классы будет являться его наследником.

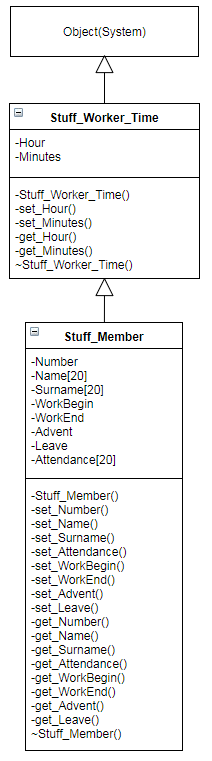


Рис.3 – Диаграмма иерархии классов

## ОПИСАНИЕ СТРУКТУРЫ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА

Разобьем исходный код программы на модули и разработаем пользовательский графический интерфейс:

1. Stuff\_Worker\_Time.h, Stuff\_Worker\_Time.cpp – класс времени прихода/ухода на/с работы.
2. Stuff\_Member.h, Stuff\_Member.cpp – класс сотрудника.
3. Sub\_Programs.h, Sub\_Programs.cpp – модуль с основными подпрограммами.
4. Interface\_Main.h, Interface\_Main.cpp – главная форма содержащая главное меню программы.
5. Interface\_Add.h, Interface\_Add.cpp – форма добавления нового сотрудника в систему.
6. Add\_Appearance.h, Add\_Appearance.cpp – форма регистрации прохода сотрудника на предприятие.
7. Violation\_Manager.h, Violation\_Manager.cpp – форма просмотра нарушений сотрудников.

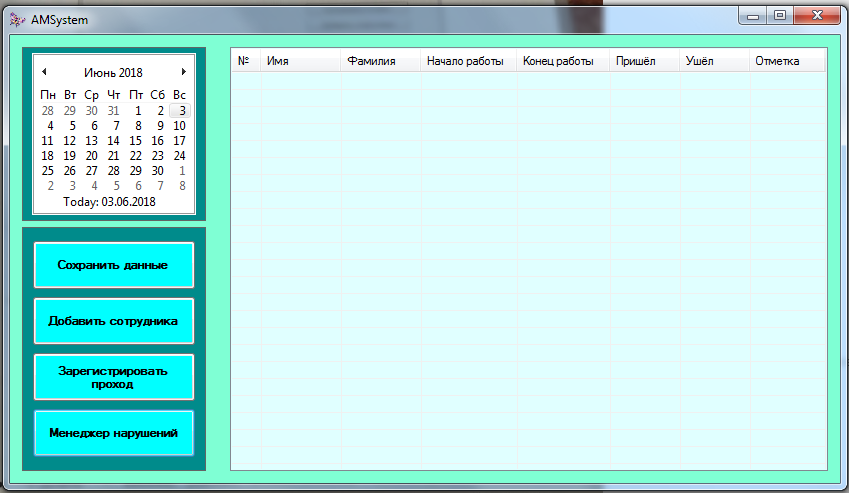


Рис. 4 – Главная форма приложения Interface\_Main

Назначение компонент главной формы приложения:

1. “Сохранить данные” – кнопка, позволяющая сохранить текущие изменения, внесенные пользователем в журнал при работе программы.
2. “Добавить сотрудника” – кнопка, вызывает форму добавления нового сотрудника в систему.
3. “Зарегистрировать проход” – кнопка, вызывает форму регистрации прохода сотрудника на предприятие.
4. “Менеджер нарушений” – кнопка, вызывающая форму управления нарушениями сотрудников.
5. “Календарь” – monthCalendar, позволяет следить за посещаемостью всех сотрудников в течение каждого дня текущего календарного года.

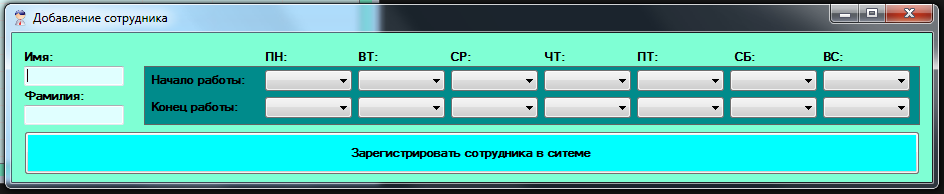


Рис. 5 – Форма добавления сотрудника Interface\_Add

Назначение компонент формы добавления сотрудника:

1. textBox1, textBox2 – поля ввода имени и фамилии сотрудника.
2. comboBox1...comboBox14 – меню составления графика работы для сотрудника.
3. “Зарегистрировать сотрудника в системе” – кнопка, позволяет добавить нового сотрудника в систему при условии корректного заполнения необходимых данных.

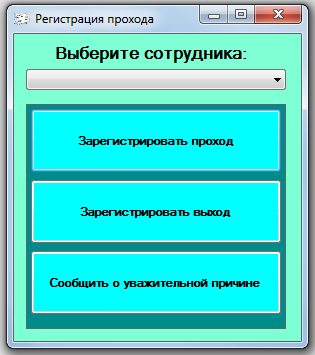


Рис. 6 – Форма регистрации прохода Add\_Appearance

Назначение компонент формы Регистрации прохода:

1. comboBox1 – комбобокс, позволяет выбрать нужного сотрудника из всего штата.
2. “Зарегистрировать проход” – кнопка, позволяет пройти на предприятие за выбранного сотрудника, если выполнены все необходимые условия.
3. “Зарегистрировать выход” – кнопка, позволяет выйти из предприятия за выбранного сотрудника, если выполнены все необходимые условия.
4. “Сообщить о уважительной причине” – кнопка, позволяет сообщить о уважительной причине для выбранного сотрудника.

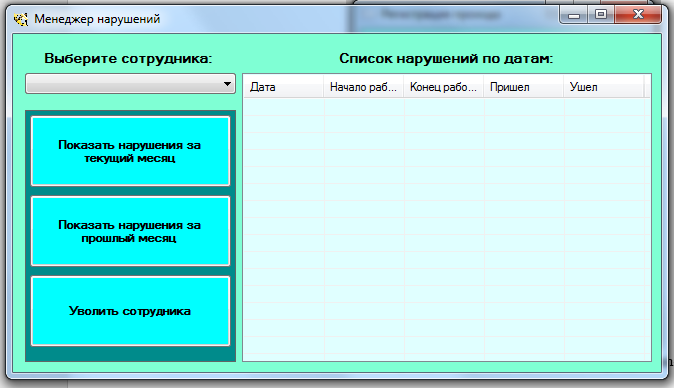


Рис. 7 – форма менеджера нарушений Violation\_Manager

Назначение компонент формы Менеджера нарушений:

1. comboBox1 – комбобокс, позволяет выбрать нужного сотрудника из всего штата.
2. “Показать нарушения за текущий месяц” – кнопка, позволяет посмотреть нарушения выбранного сотрудника за текущий месяц.
3. “Показать нарушения за прошлый месяц” – кнопка, позволяет посмотреть нарушения выбранного сотрудника за прошлый месяц.
4. “Уволить сотрудника” – кнопка, позволяет уволить выбранного сотрудника.

# ЭКСПЕРИМЕНТАЛЬНАЯ ЧАСТЬ

## ОПИСАНИЕ ПОДХОДА К ТЕСТИРОВАНИЮ

Протестируем разработанную систему, выполнив для неё следующие операции:

1. Просмотреть отчёт о посещаемости сотрудников за какой-то день прошлого года.
2. Добавить нового сотрудника в систему, указав его имя, фамилию и рабочие дни.
3. Зарегистрировать проход за одного из существующих на текущий момент в штате сотрудников.
4. Посмотреть нарушения для одного из существующих в штате сотрудников за этот месяц.
5. Посмотреть нарушения для одного из существующих в штате сотрудников за прошлый месяц.
6. Уволить одного из существующих в штате сотрудников.
7. Сохранить измененные данные системы в файл.

## ТЕСТИРОВАНИЕ ПРОГРАММЫ

Тест №1. Просмотреть отчёт о посещаемости сотрудников за какой-то день прошлого года.

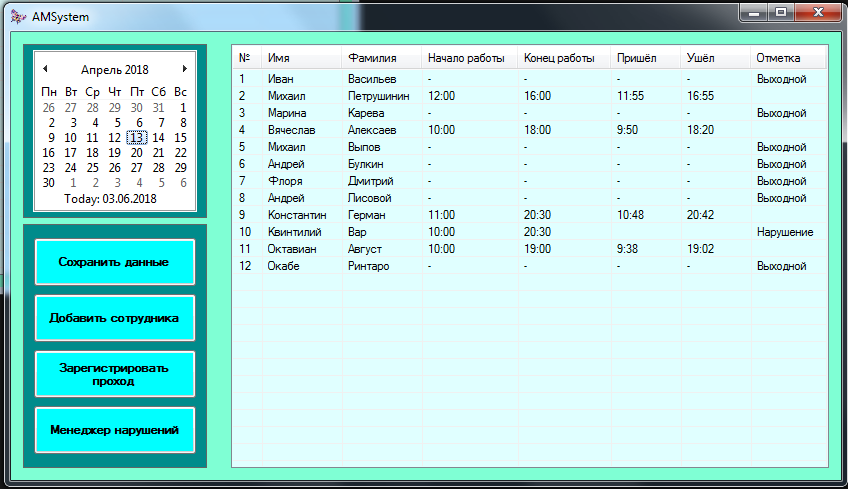


Рис. 8 – Результат теста №1.

Тест №2. Добавить нового сотрудника в систему, указав его имя, фамилию и рабочие дни.

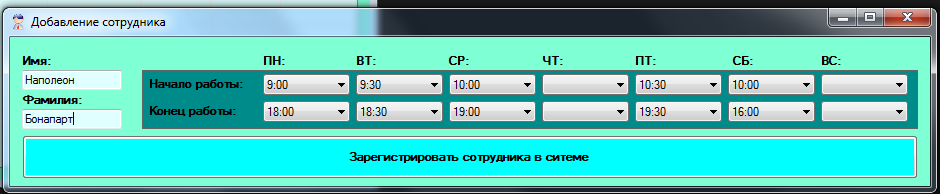


Рис. 9 – Ход теста №2.

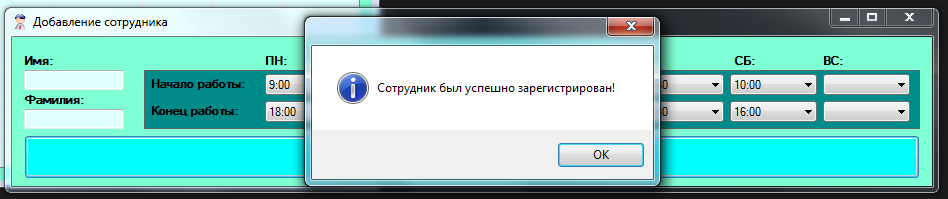


Рис. 10 – Ход теста №2.

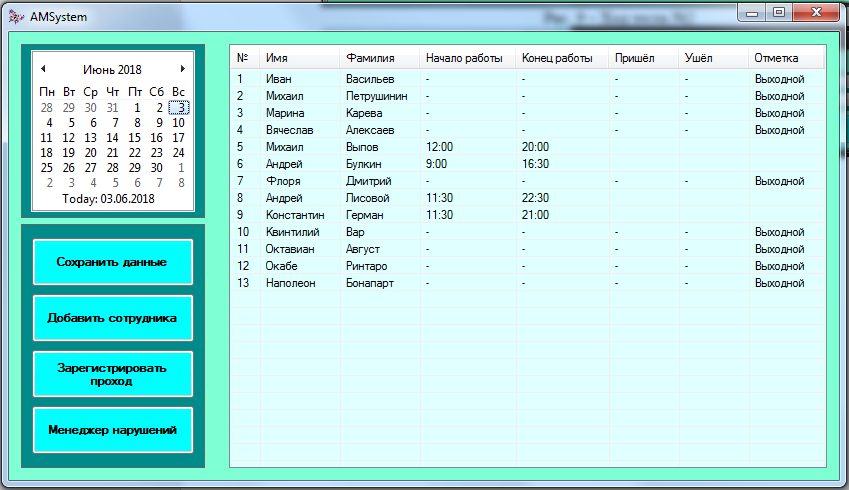


Рис. 11 – Результат теста №2.

Тест №3. Зарегистрировать проход за одного из существующих на текущий момент в штате сотрудников.

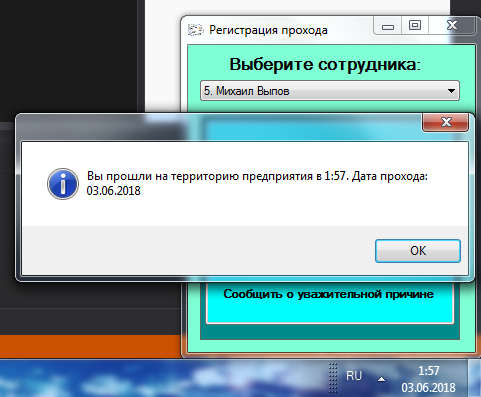


Рис. 12 – Ход теста №3.

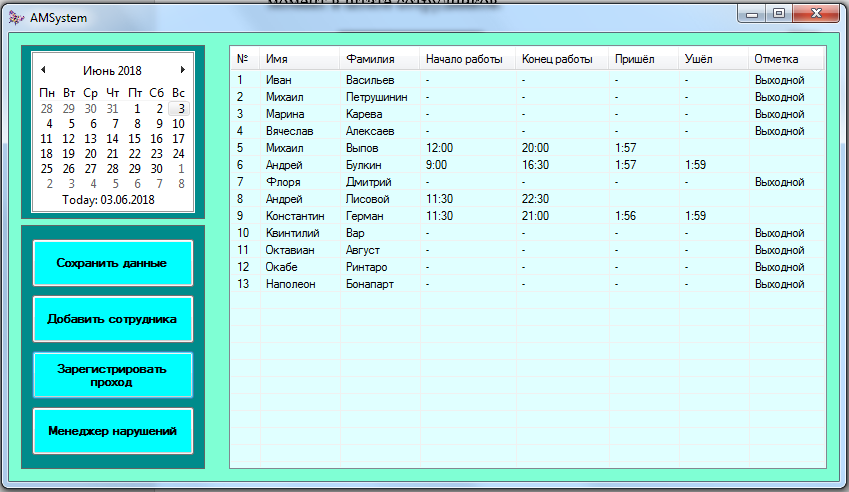


Рис. 13 – Результат теста №3.

Тест №4. Посмотреть нарушения для одного из существующих в штате сотрудников за этот месяц.

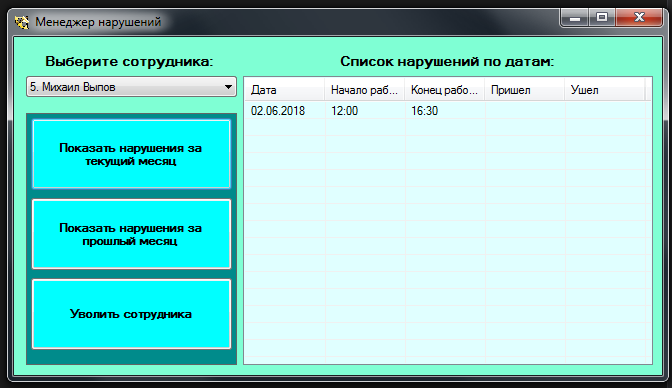


Рис. 14 – Результат теста №4, видно, что сотрудник не пришел на работу в рабочий день по его графику.

Тест №5. Посмотреть нарушения для одного из существующих в штате сотрудников за прошлый месяц.

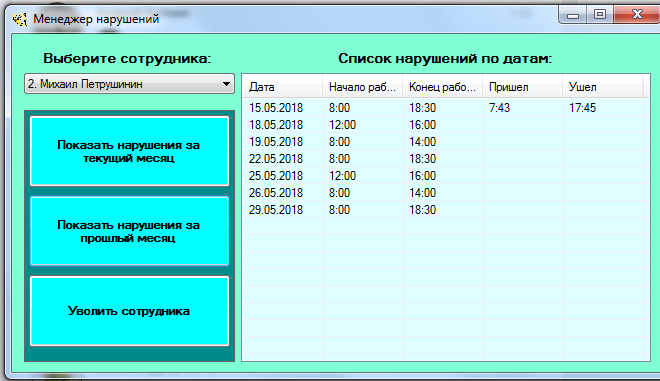


Рис. 15 – Результат теста №5.

Тест №6. Уволить одного из существующих в штате сотрудников.

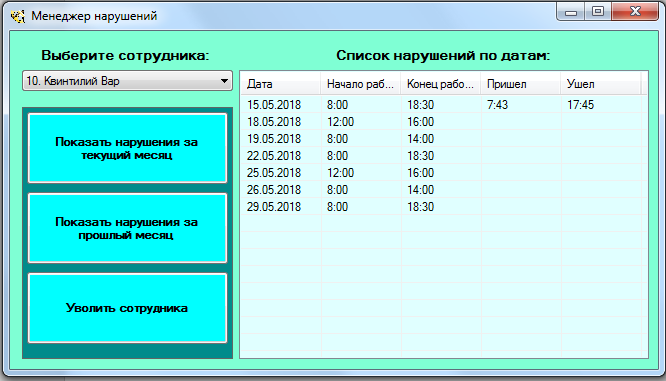


Рис. 16 – Ход тестирования №6.

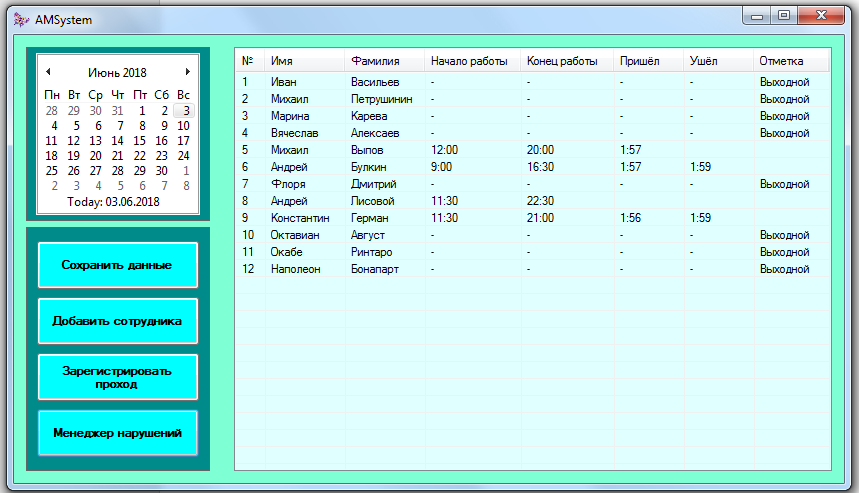


Рис. 17 – Результат тестирования №6.

Тест №7. Сохранить измененные данные системы в файл.

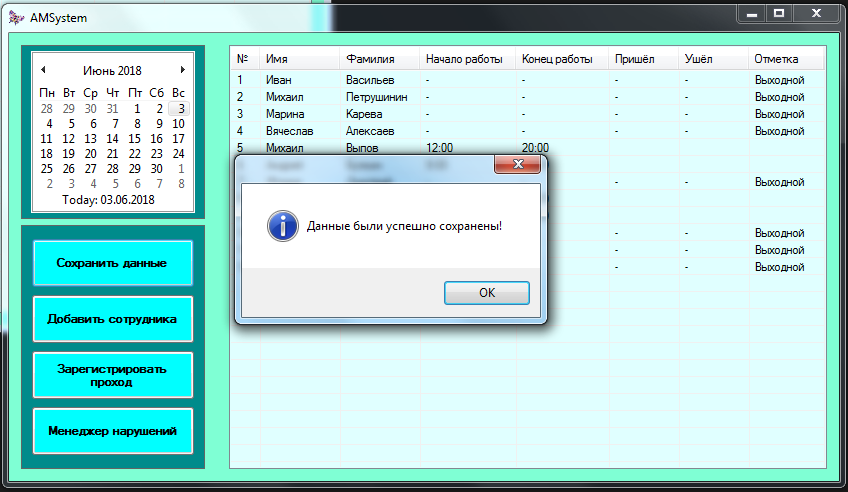


Рис. 18 – Результат тестирования №7.

## ИНСТРУКЦИЯ ПРОГРАММИСТУ

Для работы с программой желательно знать следующее:

1. Данные о посещаемости за весь текущий календарный год хранятся в одном файле AMSData.txt.
2. Загрузка данных из файла происходит автоматически при запуске программы, если файл с данными отсутствует – будет создан новый пустой файл-журнал.
3. Перед завершением работы с программой необходимо сохранить внесённые в журнал изменения.
4. При добавлении нового сотрудника его график работы будет внесен в журнал с дня добавления вплоть до конца текущего календарного года.
5. Увольнение сотрудника подразумевает также и удаление всех данных о нём из журнала.
6. При проходе сотрудника на предприятие запоминается время первого входа и последнего выхода.
7. Нарушения проставляются программой при наступлении следующего дня, при наличии необходимых оснований для этого. Уважительная причина позволяет избежать проставления нарушения в случае её заведомого наличия.

# ЗАКЛЮЧЕНИЕ

В данный момент Объектно-Ориентированное Программирование по праву считается и является одним из самых востребованных и динамично развивающихся парадигм программирования. Грамотное применение сей парадигмы позволяет достичь следующих столь необходимых для поддержания конкурентоспособности качеств ПО:

1. Уменьшенная сложность для понимания всего алгоритма программы в целом.
2. Повышение общей надежности программной системы.
3. Упрощенная схема модификации устаревших модулей программной системы.
4. Возможность многократного использования уже ранее написанных и подходящих для решения данной задачи модулей.

Именно возможность удобного комбинирования этих признаков в рамках данной парадигмы и объясняет её востребованность на современном рынке разработки ПО.

В ходе выполнения данной курсовой работы я понял основные концепции парадигмы ООП а так же развил свои навыки разработки приложений с пользовательским графическим интерфейсом для Windows в среде CLR на языке C++. Были разработаны необходимые для выполнения данного задания классы, а также ряд подпрограмм для взаимодействия с ними. После чего была составлена рабочая многомодульная программа в интегрированной среде разработки Microsoft Visual Studio 2017. Тестирование успешно показало работоспособность разработанной программной системы и соответствие её поставленной задаче.

# СПИСОК ЛИТЕРАТУРЫ

1. Бьерн Страуструп. Программирование. Принципы и практика с использованием C++, 2015;
2. Никита Культин. C/C++ в задачах и примерах, 2009;
3. Герберт Шилдт. C++. Базовый курс, 2014;
4. Никита Культин. Основы программирования в Microsoft Visual C++, 2010.

# ПРИЛОЖЕНИЯ

## Приложение 1. Модуль класса Stuff\_Worker\_Time.

Stuff\_Worker\_Time.h:

class Stuff\_Worker\_Time

{

private:

unsigned short int Hour : 5;

unsigned short int Minutes : 6;

public:

//Конструктор без параметров:

Stuff\_Worker\_Time();

//Конструктор с параметрами:

Stuff\_Worker\_Time(unsigned short int H, unsigned short int M);

//Set:

void set\_Hour(unsigned short int H);

void set\_Minutes(unsigned short int M);

//Get:

unsigned short int get\_Hour();

unsigned short int get\_Minutes();

//Деструктор

~Stuff\_Worker\_Time();

};

Stuff\_Worker\_Time.cpp:

#include "Stuff\_Worker\_Time.h"

//Конструктор без параметров:

Stuff\_Worker\_Time::Stuff\_Worker\_Time()

{

Hour = 0;

Minutes = 0;

}

//Конструктор с параметрами:

Stuff\_Worker\_Time::Stuff\_Worker\_Time(unsigned short int H, unsigned short int M)

{

Hour = H;

Minutes = M;

}

//Set:

void Stuff\_Worker\_Time::set\_Hour(unsigned short int H)

{

Hour = H;

}

void Stuff\_Worker\_Time::set\_Minutes(unsigned short int M)

{

Minutes = M;

}

//Get:

unsigned short int Stuff\_Worker\_Time::get\_Hour()

{

return Hour;

}

unsigned short int Stuff\_Worker\_Time::get\_Minutes()

{

return Minutes;

}

//Деструктор

Stuff\_Worker\_Time::~Stuff\_Worker\_Time()

{

}

## Приложение 2. Модуль класса Stuff\_Member.

Stuff\_Member.h:

#pragma once

#include "string.h"

#include "Stuff\_Worker\_Time.h"

class Stuff\_Member: private Stuff\_Worker\_Time

{

private:

unsigned int Number; //Номер в списке рабочих

char Name[20]; //Имя

char Surname[20]; // Фамилия

Stuff\_Worker\_Time WorkBegin; //Время прихода на работу по графику

Stuff\_Worker\_Time WorkEnd; //Время ухода с работы по графику

Stuff\_Worker\_Time Advent; //Реальное время прихода

Stuff\_Worker\_Time Leave; //Реальное время ухода

char Attendance[20]; //Отметка посещения/непосещения

public:

//Конструктор без параметров:

Stuff\_Member();

//Конструктор с параметрами:

Stuff\_Member(unsigned short Num, char\* N, char\* S,

unsigned short int H1, unsigned short int M1, unsigned short int H2, unsigned short int M2,

unsigned short int H3, unsigned short int M3, unsigned short int H4, unsigned short int M4, char\* A);

//Set:

void set\_Number(unsigned int N);

void set\_Name(char\* N);

void set\_Surname(char\* S);

void set\_Attendance(char\* A);

void set\_WorkBegin(unsigned int H, unsigned int M);

void set\_WorkEnd(unsigned int H, unsigned int M);

void set\_Advent(unsigned int H, unsigned int M);

void set\_Leave(unsigned int H, unsigned int M);

//Get:

unsigned int get\_Number();

char\* get\_Name();

char\* get\_Surname();

char\* get\_Attendance();

Stuff\_Worker\_Time get\_WorkBegin();

Stuff\_Worker\_Time get\_WorkEnd();

Stuff\_Worker\_Time get\_Advent();

Stuff\_Worker\_Time get\_Leave();

//Деструктор

~Stuff\_Member();

};

Stuff\_Member.cpp:

#include "Stuff\_Member.h"

#include "Stuff\_Worker\_Time.h"

//Конструктор по умолчанию:

Stuff\_Member::Stuff\_Member()

{

Number = 0;

strcpy\_s(Name, "N/A");

strcpy\_s(Surname, "N/A");

WorkBegin.set\_Hour(0);

WorkBegin.set\_Minutes(0);

WorkEnd.set\_Hour(0);

WorkEnd.set\_Minutes(0);

Advent.set\_Hour(0);

Advent.set\_Minutes(0);

Leave.set\_Hour(0);

Leave.set\_Minutes(0);

strcpy\_s(Attendance, "N/A");

}

//Конструктор с параметрами:

Stuff\_Member::Stuff\_Member(unsigned short Num, char\* N, char\* S,

unsigned short int H1, unsigned short int M1, unsigned short int H2, unsigned short int M2,

unsigned short int H3, unsigned short int M3, unsigned short int H4, unsigned short int M4, char\* A)

{

Number = Num;

strcpy\_s(Name, N);

strcpy\_s(Surname, S);

WorkBegin.set\_Hour(H1);

WorkBegin.set\_Minutes(M1);

WorkEnd.set\_Hour(H2);

WorkEnd.set\_Minutes(M2);

Advent.set\_Hour(H3);

Advent.set\_Minutes(M3);

Leave.set\_Hour(H4);

Leave.set\_Minutes(M4);

strcpy\_s(Attendance, A);

}

//Set:

void Stuff\_Member::set\_Number(unsigned int N)

{

Number = N;

}

void Stuff\_Member::set\_Name(char\* N)

{

strcpy\_s(Name, N);

}

void Stuff\_Member::set\_Surname(char\* S)

{

strcpy\_s(Surname, S);

}

void Stuff\_Member::set\_Attendance(char\* A)

{

strcpy\_s(Attendance, A);

}

void Stuff\_Member::set\_WorkBegin(unsigned int H, unsigned int M)

{

WorkBegin.set\_Hour(H);

WorkBegin.set\_Minutes(M);

}

void Stuff\_Member::set\_WorkEnd(unsigned int H, unsigned int M)

{

WorkEnd.set\_Hour(H);

WorkEnd.set\_Minutes(M);

}

void Stuff\_Member::set\_Advent(unsigned int H, unsigned int M)

{

Advent.set\_Hour(H);

Advent.set\_Minutes(M);

}

void Stuff\_Member::set\_Leave(unsigned int H, unsigned int M)

{

Leave.set\_Hour(H);

Leave.set\_Minutes(M);

}

//Get:

unsigned int Stuff\_Member::get\_Number()

{

return Number;

}

char\* Stuff\_Member::get\_Name()

{

return Name;

}

char\* Stuff\_Member::get\_Surname()

{

return Surname;

}

char\* Stuff\_Member::get\_Attendance()

{

return Attendance;

}

Stuff\_Worker\_Time Stuff\_Member::get\_WorkBegin()

{

return WorkBegin;

}

Stuff\_Worker\_Time Stuff\_Member::get\_WorkEnd()

{

return WorkEnd;

}

Stuff\_Worker\_Time Stuff\_Member::get\_Advent()

{

return Advent;

}

Stuff\_Worker\_Time Stuff\_Member::get\_Leave()

{

return Leave;

}

//Деструктор

Stuff\_Member::~Stuff\_Member()

{

}

## Приложение 3. Модуль подпрограмм Sub\_Programs:

Sub\_Programs.h:

#pragma once

#include <vector>

#include <list>

#include <fstream>

#include <sstream>

#include "string.h"

#include "Stuff\_Member.h"

#include <conio.h>

#include <time.h>

using namespace System::Runtime::InteropServices;

//Прототипы:

std::vector<std::vector<Stuff\_Member>> Load\_Data(std::vector<std::vector<Stuff\_Member>> Main\_Days);

std::vector<std::vector<Stuff\_Member>> Create\_Empty\_Data(std::vector<std::vector<Stuff\_Member>> Main\_Days);

std::vector<std::string> String\_Split(std::vector<std::string> v\_Worker, std::string s\_Worker);

void Save\_Data(std::vector<std::vector<Stuff\_Member>> Main\_Days);

std::vector<std::vector<Stuff\_Member>> Add\_Stuff\_Member(std::vector<std::vector<Stuff\_Member>> Main\_Days, System::String^ N, System::String^ S, System::String^ W\_B, System::String^ W\_E, unsigned int Day\_Of\_Week, unsigned int Current\_Day);

Stuff\_Worker\_Time Add\_Time(char\* Time, Stuff\_Worker\_Time W\_Time);

std::vector<std::vector<Stuff\_Member>> Add\_Marks(std::vector<std::vector<Stuff\_Member>> Main\_Days, unsigned int Current\_Day);

std::string DateFromDayNumber(int DayNumber, int Year, std::vector<int> MonthDays, std::vector<int> MonthDaysLeap);

std::string DateFromDayNumberForLastMonth(int DayNumber, int Year, std::vector<int> MonthDays, std::vector<int> MonthDaysLeap);

std::vector<std::vector<Stuff\_Member>> Dismiss\_Stuff\_Member(std::vector<std::vector<Stuff\_Member>> Main\_Days, unsigned int Current\_Day, unsigned int Current\_Number);

Sub\_Programs.cpp:

#include "Sub\_Programs.h"

//Загрузить данные:

std::vector<std::vector<Stuff\_Member>> Load\_Data(std::vector<std::vector<Stuff\_Member>> Main\_Days)

{

char Worker\_Controller[10];

int i = -1;

std::ifstream fin;

std::stringstream ss\_Worker;

std::string s\_Worker;

fin.open("AMS\_Data.txt");

if (!fin.is\_open())

{

std::ofstream ofs("AMS\_Data.txt");

Main\_Days = Create\_Empty\_Data(Main\_Days);

Save\_Data(Main\_Days);

ofs.close();

return Main\_Days;

}

while (true)

{

int Worker\_Controller\_Count = 0;

while (i + 1 <= Main\_Days.size())

{

getline(fin, s\_Worker);

ss\_Worker.str(s\_Worker);

ss\_Worker >> Worker\_Controller;

if (strcmp(Worker\_Controller, "Workers,") == 0)

Worker\_Controller\_Count++;

if (Worker\_Controller\_Count == 1)

{

i++;

break;

}

else

{

//Вектор под выцепленные слова из строки файла

std::vector<std::string> v\_Worker;

//Процедура заполнения вектора лексемами

v\_Worker = String\_Split(v\_Worker, s\_Worker);

//буферное Имя из файла

char\* Temp\_Name = new char[v\_Worker[1].size() + 1];

std::copy(v\_Worker[1].begin(), v\_Worker[1].end(), Temp\_Name);

Temp\_Name[v\_Worker[1].size()] = '\0';

//буферная Фамилия из файла

char\* Temp\_Surname = new char[v\_Worker[2].size() + 1];

std::copy(v\_Worker[2].begin(), v\_Worker[2].end(), Temp\_Surname);

Temp\_Surname[v\_Worker[2].size()] = '\0';

//буферная Отметка из файла

char\* Temp\_Attendance = new char[v\_Worker[11].size() + 1];

std::copy(v\_Worker[11].begin(), v\_Worker[11].end(), Temp\_Attendance);

Temp\_Attendance[v\_Worker[11].size()] = '\0';

//Создадим буферный экземлпяр класса Stuff\_Member, используя конструктор с параметрами и данные из файла

Stuff\_Member S\_M = Stuff\_Member

(

std::atoi(v\_Worker[0].c\_str()),

Temp\_Name, //Имя

Temp\_Surname, //Фамилия

std::atoi(v\_Worker[3].c\_str()), std::atoi(v\_Worker[4].c\_str()), //Начало работы по графику

std::atoi(v\_Worker[5].c\_str()), std::atoi(v\_Worker[6].c\_str()), //Конец работы по графику

std::atoi(v\_Worker[7].c\_str()), std::atoi(v\_Worker[8].c\_str()), //Реальный приход

std::atoi(v\_Worker[9].c\_str()), std::atoi(v\_Worker[10].c\_str()), //Реальный уход

Temp\_Attendance //Отметка

);

Main\_Days[i].push\_back(S\_M);

delete[] Temp\_Name;

delete[] Temp\_Surname;

delete[] Temp\_Attendance;

}

}

//Выйти из цикла перебора списков рабочих

if (i == 365)

{

fin.close();

return Main\_Days;

break;

}

}

}

//Конструктор пустого файла:

std::vector<std::vector<Stuff\_Member>> Create\_Empty\_Data(std::vector<std::vector<Stuff\_Member>> Main\_Days)

{

Stuff\_Member S\_F = Stuff\_Member();

for (int i = 0; i < Main\_Days.size(); i++)

{

Main\_Days[i].push\_back(S\_F);

}

return Main\_Days;

}

//Разбить строку файла на лексемы:

std::vector<std::string> String\_Split(std::vector<std::string> v\_Worker, std::string s\_Worker)

{

//Перевод строки в сишный вид

char\* s\_Temp = new char[s\_Worker.size() + 1];

std::copy(s\_Worker.begin(), s\_Worker.end(), s\_Temp);

s\_Temp[s\_Worker.size()] = '\0';

//Переменная, в которую будут заноситься начальные адреса частей строки

char \*s\_Pointer;

//Выделение первой части строки

s\_Pointer = strtok(s\_Temp, " ");

//Выделение последующих частей

while (s\_Pointer != NULL)

{

v\_Worker.push\_back(s\_Pointer);

s\_Pointer = strtok(NULL, " ");

}

return v\_Worker;

}

//Сохранить данные:

void Save\_Data(std::vector<std::vector<Stuff\_Member>> Main\_Days)

{

std::ofstream fout("AMS\_Data.txt");

if (!fout.is\_open())

{

std::ofstream ofs("AMS\_Data.txt");

ofs.close();

}

for (int i = 0; i < Main\_Days.size(); i++)

{

fout << "Workers," << " Day: " << i + 1 << std::endl;

for (int j = 0; j < Main\_Days[i].size(); j++)

{

fout //Запись дня с списком рабочих осуществляется как список строк, где каждая строка - один работник и его данные

<< Main\_Days[i][j].get\_Number() << " "

<< Main\_Days[i][j].get\_Name() << " "

<< Main\_Days[i][j].get\_Surname() << " "

<< Main\_Days[i][j].get\_WorkBegin().get\_Hour() << " " << Main\_Days[i][j].get\_WorkBegin().get\_Minutes() << " "

<< Main\_Days[i][j].get\_WorkEnd().get\_Hour() << " " << Main\_Days[i][j].get\_WorkEnd().get\_Minutes() << " "

<< Main\_Days[i][j].get\_Advent().get\_Hour() << " " << Main\_Days[i][j].get\_Advent().get\_Minutes() << " "

<< Main\_Days[i][j].get\_Leave().get\_Hour() << " " << Main\_Days[i][j].get\_Leave().get\_Minutes() << " "

<< Main\_Days[i][j].get\_Attendance() << std::endl;

}

}

fout << "Workers, the end of year." << std::endl;

fout.close();

}

//Добавить сотрудника:

std::vector<std::vector<Stuff\_Member>> Add\_Stuff\_Member(std::vector<std::vector<Stuff\_Member>> Main\_Days, System::String^ N, System::String^ S, System::String^ W\_B, System::String^ W\_E, unsigned int Day\_Of\_Week, unsigned int Current\_Day)

{

//Преобразование типов строк

char\* Name = (char\*)(void\*)Marshal::StringToHGlobalAnsi(N);

char\* Surname = (char\*)(void\*)Marshal::StringToHGlobalAnsi(S);

char\* Work\_Begin = (char\*)(void\*)Marshal::StringToHGlobalAnsi(W\_B);

char\* Work\_End = (char\*)(void\*)Marshal::StringToHGlobalAnsi(W\_E);

//Выделим время

Stuff\_Worker\_Time W\_B\_Time;

Stuff\_Worker\_Time W\_E\_Time;

if (!((W\_B == "") || (W\_E == ""))) //Условие непустого времени

{

W\_B\_Time = Add\_Time(Work\_Begin, W\_B\_Time);

W\_E\_Time = Add\_Time(Work\_End, W\_E\_Time);

}

else

{

W\_B\_Time.set\_Hour(0);

W\_B\_Time.set\_Minutes(0);

W\_E\_Time.set\_Hour(0);

W\_E\_Time.set\_Minutes(0);

}

//Запишем в нашу структуру данных расписание работника в этот день недели начиная со дня добавления и до конца года:

//Исключим выходные для стандартного конструктора:

if ((W\_B != "") && (W\_E != ""))

{

for (int i = Day\_Of\_Week - 1; i < Main\_Days.size(); i += 7)

{

//Создадим экземпляр класса работника, куда поместим полученные данные, использовав конструктор с параметрами:

Stuff\_Member S\_F = Stuff\_Member(Main\_Days[i].size(), Name, Surname, W\_B\_Time.get\_Hour(), W\_B\_Time.get\_Minutes(), W\_E\_Time.get\_Hour(), W\_E\_Time.get\_Minutes(), 0, 0, 0, 0, "N/A");

if (i >= Current\_Day - 1)

{

Main\_Days[i].push\_back(S\_F);

}

}

}

//Заполним данные для выходных через другие параметры конструктора с параметрами для класса работник:

else if ((W\_B == "") || (W\_E == ""))

{

for (int i = Day\_Of\_Week - 1; i < Main\_Days.size(); i += 7)

{

//Создадим экземпляр класса работника, куда поместим полученные данные, использовав конструктор с параметрами:

Stuff\_Member S\_F = Stuff\_Member(Main\_Days[i].size(), Name, Surname, 0, 0, 0, 0, 0, 0, 0, 0, "N/A");

if (i >= Current\_Day - 1)

{

Main\_Days[i].push\_back(S\_F);

}

}

}

return Main\_Days;

}

//Выделить время:

Stuff\_Worker\_Time Add\_Time(char\* Time, Stuff\_Worker\_Time W\_Time)

{

char \*s\_Pointer;

std::vector<unsigned short int> Time\_Buffer;

//Разибитие строки на токены и загон их в буфер с приведением типов

s\_Pointer = strtok(Time, ":");

while (s\_Pointer != NULL)

{

Time\_Buffer.push\_back(atoi(s\_Pointer));

s\_Pointer = strtok(NULL, ":");

}

//Перенос из буфера в поля класса

W\_Time.set\_Hour(Time\_Buffer[0]);

W\_Time.set\_Minutes(Time\_Buffer[1]);

return W\_Time;

}

//Проставить отметки:

std::vector<std::vector<Stuff\_Member>> Add\_Marks(std::vector<std::vector<Stuff\_Member>> Main\_Days, unsigned int Current\_Day)

{

for (int i = 0; i < Main\_Days.size(); i++)

{

for (int j = 0; j < Main\_Days[i].size(); j++)

{

if ((i < Current\_Day - 1) &&

(Main\_Days[i][j].get\_Advent().get\_Hour() + Main\_Days[i][j].get\_Advent().get\_Minutes()) != 0 &&

(Main\_Days[i][j].get\_Leave().get\_Hour() + Main\_Days[i][j].get\_Leave().get\_Minutes()) == 0)

{

Main\_Days[i][j].set\_Leave(23, 59); //Обозначим время закрытия предприятия, на случай, если сотрудник не ушел с работы сам, как 23:59

}

else if (Main\_Days[i][j].get\_WorkBegin().get\_Hour() == 0 && Main\_Days[i][j].get\_WorkBegin().get\_Minutes() == 0

&& Main\_Days[i][j].get\_WorkEnd().get\_Hour() == 0 && Main\_Days[i][j].get\_WorkEnd().get\_Minutes() == 0)

{

Main\_Days[i][j].set\_Attendance("В");

}

else if ((i < Current\_Day - 1) &&

(((Main\_Days[i][j].get\_WorkBegin().get\_Hour() < Main\_Days[i][j].get\_Advent().get\_Hour()) ||

((Main\_Days[i][j].get\_WorkBegin().get\_Minutes() < Main\_Days[i][j].get\_Advent().get\_Minutes()) && (Main\_Days[i][j].get\_WorkBegin().get\_Hour() == Main\_Days[i][j].get\_Advent().get\_Hour()))) ||

((Main\_Days[i][j].get\_WorkEnd().get\_Hour() > Main\_Days[i][j].get\_Leave().get\_Hour()) ||

((Main\_Days[i][j].get\_WorkEnd().get\_Minutes() > Main\_Days[i][j].get\_Leave().get\_Minutes()) && (Main\_Days[i][j].get\_WorkEnd().get\_Hour() == Main\_Days[i][j].get\_Leave().get\_Hour()))) &&

(strcmp(Main\_Days[i][j].get\_Attendance(), "У") != 0)))

{

Main\_Days[i][j].set\_Attendance("Н");

}

else

{

if (strcmp(Main\_Days[i][j].get\_Attendance(), "У") != 0)

Main\_Days[i][j].set\_Attendance("П");

}

}

}

return Main\_Days;

}

//Получить дату из номера дня для текущего месяца:

std::string DateFromDayNumber(int DayNumber, int Year, std::vector<int> MonthDays, std::vector<int> MonthDaysLeap)

{

//Текущее время

time\_t t;

struct tm \*t\_m;

t = time(NULL);

t\_m = localtime(&t);

//Количество дней для выделения номера дня в месяце

int FullCountOfDays = 0;

//Номер дня в месяце

int DayNumberInMonth = 0;

//Номер месяца

int NumberOfMonth = 0;

//Номер года

int NumberOfYear = t\_m->tm\_year + 1900;

//Итоговая дата

std::string Date;

std::string Day, Month, SYear;

//На случай обычного года:

if (Year == 365)

{

for (int i = 0; i < t\_m->tm\_mon; i++)

{

FullCountOfDays += MonthDays[i];

NumberOfMonth++;

}

//Вычислим номер дня в месяце

DayNumberInMonth = DayNumber - FullCountOfDays;

//Преобразуем все в строку:

Day = std::to\_string(DayNumberInMonth + 1);

Month = std::to\_string(NumberOfMonth + 1);

SYear = std::to\_string(NumberOfYear);

if (DayNumberInMonth < 9 && NumberOfMonth < 9)

{

Date = "0" + Day + ".0" + Month + "." + SYear;

}

else if (NumberOfMonth < 9)

{

Date = Day + ".0" + Month + "." + SYear;

}

else if (DayNumberInMonth < 9)

{

Date = "0" + Day + "." + Month + "." + SYear;

}

else

{

Date = Day + "." + Month + "." + SYear;

}

}

//На случай високосного года

else

{

for (int i = 0; i < t\_m->tm\_mon; i++)

{

FullCountOfDays += MonthDaysLeap[i];

NumberOfMonth++;

}

//Вычислим номер дня в месяце

DayNumberInMonth = DayNumber - FullCountOfDays;

//Преобразуем все в строку:

Day = std::to\_string(DayNumberInMonth + 1);

Month = std::to\_string(NumberOfMonth + 1);

SYear = std::to\_string(NumberOfYear);

if (DayNumberInMonth < 9 && NumberOfMonth < 9)

{

Date = "0" + Day + ".0" + Month + "." + SYear;

}

else if (NumberOfMonth < 9)

{

Date = Day + ".0" + Month + "." + SYear;

}

else if (DayNumberInMonth < 9)

{

Date = "0" + Day + "." + Month + "." + SYear;

}

else

{

Date = Day + "." + Month + "." + SYear;

}

}

return Date;

}

//Получить дату из номера дня для предыдущего месяца:

std::string DateFromDayNumberForLastMonth(int DayNumber, int Year, std::vector<int> MonthDays, std::vector<int> MonthDaysLeap)

{

//Текущее время

time\_t t;

struct tm \*t\_m;

t = time(NULL);

t\_m = localtime(&t);

//Количество дней для выделения номера дня в месяце

int FullCountOfDays = 0;

//Номер дня в месяце

int DayNumberInMonth = 0;

//Номер месяца

int NumberOfMonth = 0;

//Номер года

int NumberOfYear = t\_m->tm\_year + 1900;

//Итоговая дата

std::string Date;

std::string Day, Month, SYear;

//На случай обычного года:

if (Year == 365)

{

for (int i = 0; i < t\_m->tm\_mon - 1; i++)

{

FullCountOfDays += MonthDays[i];

NumberOfMonth++;

}

//Вычислим номер дня в месяце

DayNumberInMonth = DayNumber - FullCountOfDays;

//Преобразуем все в строку:

Day = std::to\_string(DayNumberInMonth + 1);

Month = std::to\_string(NumberOfMonth + 1);

SYear = std::to\_string(NumberOfYear);

if (DayNumberInMonth < 9 && NumberOfMonth < 9)

{

Date = "0" + Day + ".0" + Month + "." + SYear;

}

else if (NumberOfMonth < 9)

{

Date = Day + ".0" + Month + "." + SYear;

}

else if (DayNumberInMonth < 9)

{

Date = "0" + Day + "." + Month + "." + SYear;

}

else

{

Date = Day + "." + Month + "." + SYear;

}

}

//На случай високосного года

else

{

for (int i = 0; i < t\_m->tm\_mon - 1; i++)

{

FullCountOfDays += MonthDaysLeap[i];

NumberOfMonth++;

}

//Вычислим номер дня в месяце

DayNumberInMonth = DayNumber - FullCountOfDays;

//Преобразуем все в строку:

Day = std::to\_string(DayNumberInMonth + 1);

Month = std::to\_string(NumberOfMonth + 1);

SYear = std::to\_string(NumberOfYear);

if (DayNumberInMonth < 9 && NumberOfMonth < 9)

{

Date = "0" + Day + ".0" + Month + "." + SYear;

}

else if (NumberOfMonth < 9)

{

Date = Day + ".0" + Month + "." + SYear;

}

else if (DayNumberInMonth < 9)

{

Date = "0" + Day + "." + Month + "." + SYear;

}

else

{

Date = Day + "." + Month + "." + SYear;

}

}

return Date;

}

//Уволить сотрудника, удалив его данные из системы:

std::vector<std::vector<Stuff\_Member>> Dismiss\_Stuff\_Member(std::vector<std::vector<Stuff\_Member>> Main\_Days, unsigned int Current\_Day, unsigned int Current\_Number)

{

for (int i = 0; i < Main\_Days.size(); i++)

{

if (Current\_Number < Main\_Days[i].size())

{

Main\_Days[i].erase(Main\_Days[i].begin() + Current\_Number);

for (int j = Current\_Number; j < Main\_Days[i].size(); j++)

{

//Уменьшим номера вышестоящих в списке на единицу:

if (j >= Current\_Number)

{

Main\_Days[i][j].set\_Number(Main\_Days[i][j].get\_Number() - 1);

}

}

}

}

return Main\_Days;

}

## Приложение 4. Главная форма Interface\_Main:

Sub\_Programs.cpp:

#include "Interface\_Main.h"

#include "Interface\_Add.h"

#include "Add\_Appearance.h"

#include "Violation\_Manager.h"

using namespace System;

using namespace System::Windows::Forms;

[STAThread] void Main(array<String^>^ arg)

{

Application::EnableVisualStyles();

Application::SetCompatibleTextRenderingDefault(false);

AMSystem::Interface\_Main InterfaceMain;

Application::Run(%InterfaceMain);

}

//Объявим переменную-флажок, которая будет определять состояние входа и выхода:

bool flag = false;

//Опередлим переменные для хранения и обработки данных программы

unsigned int Default\_Year = 365;

unsigned int Leap\_Year = 366;

unsigned short int Current\_Day = 0;

unsigned short int Changed\_Day = 0;

std::vector<std::vector<Stuff\_Member>> Main\_Days(Default\_Year); //Список дней с посещаемостью всех сотрудников

//Вектора с индексами-месяцами и кол-вом дней в них:

std::vector<int> MonthDays{ 31,28,31,30,31,30,31,31,30,31,30,31 };

std::vector<int> MonthDaysLeap{ 31,29,31,30,31,30,31,31,30,31,30,31 };

//Пусть только прототипы обработчиков событий останутся в заголовке, сами функции определим здесь

namespace AMSystem

{

System::Void Interface\_Main::Interface\_Main\_Load(System::Object^ sender, System::EventArgs^ e)

{

//Вернем день текущей даты:

Current\_Day = monthCalendar1->TodayDate.DayOfYear;

//Загрузка данных осуществляется с загрузкой самой формы

Main\_Days = Load\_Data(Main\_Days);

//Проставим нарушения сотрудникам за прошлые дни, если таковые имеются:

Main\_Days = Add\_Marks(Main\_Days, Current\_Day);

//Отобразим список рабочих текущего дня:

//Зададим цикл вывода списка для текущего(базового при загрузке) дня:

if (Main\_Days[Current\_Day - 1].size() != 1)

{

for (int i = 0; i < Main\_Days[Current\_Day - 1].size() - 1; i++)

{

if (Main\_Days[Current\_Day - 1][i + 1].get\_WorkBegin().get\_Hour() == 0 && Main\_Days[Current\_Day - 1][i + 1].get\_WorkBegin().get\_Minutes() == 0

&& Main\_Days[Current\_Day - 1][i + 1].get\_WorkEnd().get\_Hour() == 0 && Main\_Days[Current\_Day - 1][i + 1].get\_WorkEnd().get\_Minutes() == 0)

{

//Вывод для выходного дня:

listView1->Items->Add(Main\_Days[Current\_Day - 1][i + 1].get\_Number().ToString()); //Добавить поле Номер в первую колонку, далее последовательно привязываем другие поля экземпляра класса

listView1->Items[i]->SubItems->Add(gcnew System::String(Main\_Days[Current\_Day - 1][i + 1].get\_Name()));

listView1->Items[i]->SubItems->Add(gcnew System::String(Main\_Days[Current\_Day - 1][i + 1].get\_Surname()));

//Приход

listView1->Items[i]->SubItems->Add("-");

//Уход

listView1->Items[i]->SubItems->Add("-");

//Реальный приход

listView1->Items[i]->SubItems->Add("-");

//Реальный уход

listView1->Items[i]->SubItems->Add("-");

//Посещаемость

listView1->Items[i]->SubItems->Add("Выходной");

}

else

{

//Вывод для рабочего дня:

listView1->Items->Add(Main\_Days[Current\_Day - 1][i + 1].get\_Number().ToString()); //Добавить поле Номер в первую колонку, далее последовательно привязываем другие поля экземпляра класса

listView1->Items[i]->SubItems->Add(gcnew System::String(Main\_Days[Current\_Day - 1][i + 1].get\_Name()));

listView1->Items[i]->SubItems->Add(gcnew System::String(Main\_Days[Current\_Day - 1][i + 1].get\_Surname()));

//Приход

if (Main\_Days[Current\_Day - 1][i + 1].get\_WorkBegin().get\_Minutes() < 10)

listView1->Items[i]->SubItems->Add(Main\_Days[Current\_Day - 1][i + 1].get\_WorkBegin().get\_Hour().ToString() + ":" + "0" + Main\_Days[Current\_Day - 1][i + 1].get\_WorkBegin().get\_Minutes().ToString());

else

listView1->Items[i]->SubItems->Add(Main\_Days[Current\_Day - 1][i + 1].get\_WorkBegin().get\_Hour().ToString() + ":" + Main\_Days[Current\_Day - 1][i + 1].get\_WorkBegin().get\_Minutes().ToString());

//Уход

if (Main\_Days[Current\_Day - 1][i + 1].get\_WorkEnd().get\_Minutes() < 10)

listView1->Items[i]->SubItems->Add(Main\_Days[Current\_Day - 1][i + 1].get\_WorkEnd().get\_Hour().ToString() + ":" + "0" + Main\_Days[Current\_Day - 1][i + 1].get\_WorkEnd().get\_Minutes().ToString());

else

listView1->Items[i]->SubItems->Add(Main\_Days[Current\_Day - 1][i + 1].get\_WorkEnd().get\_Hour().ToString() + ":" + Main\_Days[Current\_Day - 1][i + 1].get\_WorkEnd().get\_Minutes().ToString());

//Реальный приход

if (Main\_Days[Current\_Day - 1][i + 1].get\_Advent().get\_Hour() == 0 && Main\_Days[Current\_Day - 1][i + 1].get\_WorkBegin().get\_Hour() != 0)

listView1->Items[i]->SubItems->Add("");

else if (Main\_Days[Current\_Day - 1][i + 1].get\_Advent().get\_Minutes() < 10)

listView1->Items[i]->SubItems->Add(Main\_Days[Current\_Day - 1][i + 1].get\_Advent().get\_Hour().ToString() + ":" + "0" + Main\_Days[Current\_Day - 1][i + 1].get\_Advent().get\_Minutes().ToString());

else

listView1->Items[i]->SubItems->Add(Main\_Days[Current\_Day - 1][i + 1].get\_Advent().get\_Hour().ToString() + ":" + Main\_Days[Current\_Day - 1][i + 1].get\_Advent().get\_Minutes().ToString());

//Реальный уход

if (Main\_Days[Current\_Day - 1][i + 1].get\_Leave().get\_Hour() == 0 && Main\_Days[Current\_Day - 1][i + 1].get\_WorkEnd().get\_Hour() != 0)

listView1->Items[i]->SubItems->Add("");

else if (Main\_Days[Current\_Day - 1][i + 1].get\_Leave().get\_Minutes() < 10)

listView1->Items[i]->SubItems->Add(Main\_Days[Current\_Day - 1][i + 1].get\_Leave().get\_Hour().ToString() + ":" + "0" + Main\_Days[Current\_Day - 1][i + 1].get\_Leave().get\_Minutes().ToString());

else

listView1->Items[i]->SubItems->Add(Main\_Days[Current\_Day - 1][i + 1].get\_Leave().get\_Hour().ToString() + ":" + Main\_Days[Current\_Day - 1][i + 1].get\_Leave().get\_Minutes().ToString());

if (strcmp(Main\_Days[Current\_Day - 1][i + 1].get\_Attendance(), "П") == 0)

{

listView1->Items[i]->SubItems->Add("");

}

else if (strcmp(Main\_Days[Current\_Day - 1][i + 1].get\_Attendance(), "Н") == 0)

{

listView1->Items[i]->SubItems->Add("Нарушение");

}

else if (strcmp(Main\_Days[Current\_Day - 1][i + 1].get\_Attendance(), "У") == 0)

{

listView1->Items[i]->SubItems->Add("Ув. Причина");

}

}

}

}

}

System::Void Interface\_Main::button1\_Click(System::Object^ sender, System::EventArgs^ e)

{

Save\_Data(Main\_Days);

MessageBox::Show("Данные были успешно сохранены!", "", MessageBoxButtons::OK, MessageBoxIcon::Asterisk);

}

System::Void Interface\_Main::button2\_Click(System::Object^ sender, System::EventArgs^ e)

{

Interface\_Add^ InterfaceAdd = gcnew Interface\_Add(this);

this->Hide();

InterfaceAdd->ShowDialog();

this->Show();

//Обновим таблицу в главной форме после операции добавления сотрудника:

//Очистим список от предыдущего списка рабочих:

for (int i = listView1->Items->Count - 1; i >= 0; i--)

{

listView1->Items[i]->Remove();

}

//Отобразим список рабочих текущего дня:

//Зададим цикл вывода списка для текущего(базового при загрузке) дня:

if (Main\_Days[Current\_Day - 1].size() != 1)

{

for (int i = 0; i < Main\_Days[Current\_Day - 1].size() - 1; i++)

{

if (Main\_Days[Current\_Day - 1][i + 1].get\_WorkBegin().get\_Hour() == 0 && Main\_Days[Current\_Day - 1][i + 1].get\_WorkBegin().get\_Minutes() == 0

&& Main\_Days[Current\_Day - 1][i + 1].get\_WorkEnd().get\_Hour() == 0 && Main\_Days[Current\_Day - 1][i + 1].get\_WorkEnd().get\_Minutes() == 0)

{

//Вывод для выходного дня:

listView1->Items->Add(Main\_Days[Current\_Day - 1][i + 1].get\_Number().ToString()); //Добавить поле Номер в первую колонку, далее последовательно привязываем другие поля экземпляра класса

listView1->Items[i]->SubItems->Add(gcnew System::String(Main\_Days[Current\_Day - 1][i + 1].get\_Name()));

listView1->Items[i]->SubItems->Add(gcnew System::String(Main\_Days[Current\_Day - 1][i + 1].get\_Surname()));

//Приход

listView1->Items[i]->SubItems->Add("-");

//Уход

listView1->Items[i]->SubItems->Add("-");

//Реальный приход

listView1->Items[i]->SubItems->Add("-");

//Реальный уход

listView1->Items[i]->SubItems->Add("-");

//Посещаемость

listView1->Items[i]->SubItems->Add("Выходной");

}

else

{

//Вывод для рабочего дня:

listView1->Items->Add(Main\_Days[Current\_Day - 1][i + 1].get\_Number().ToString()); //Добавить поле Номер в первую колонку, далее последовательно привязываем другие поля экземпляра класса

listView1->Items[i]->SubItems->Add(gcnew System::String(Main\_Days[Current\_Day - 1][i + 1].get\_Name()));

listView1->Items[i]->SubItems->Add(gcnew System::String(Main\_Days[Current\_Day - 1][i + 1].get\_Surname()));

//Приход

if (Main\_Days[Current\_Day - 1][i + 1].get\_WorkBegin().get\_Minutes() < 10)

listView1->Items[i]->SubItems->Add(Main\_Days[Current\_Day - 1][i + 1].get\_WorkBegin().get\_Hour().ToString() + ":" + "0" + Main\_Days[Current\_Day - 1][i + 1].get\_WorkBegin().get\_Minutes().ToString());

else

listView1->Items[i]->SubItems->Add(Main\_Days[Current\_Day - 1][i + 1].get\_WorkBegin().get\_Hour().ToString() + ":" + Main\_Days[Current\_Day - 1][i + 1].get\_WorkBegin().get\_Minutes().ToString());

//Уход

if (Main\_Days[Current\_Day - 1][i + 1].get\_WorkEnd().get\_Minutes() < 10)

listView1->Items[i]->SubItems->Add(Main\_Days[Current\_Day - 1][i + 1].get\_WorkEnd().get\_Hour().ToString() + ":" + "0" + Main\_Days[Current\_Day - 1][i + 1].get\_WorkEnd().get\_Minutes().ToString());

else

listView1->Items[i]->SubItems->Add(Main\_Days[Current\_Day - 1][i + 1].get\_WorkEnd().get\_Hour().ToString() + ":" + Main\_Days[Current\_Day - 1][i + 1].get\_WorkEnd().get\_Minutes().ToString());

//Реальный приход

if (Main\_Days[Current\_Day - 1][i + 1].get\_Advent().get\_Hour() == 0 && Main\_Days[Current\_Day - 1][i + 1].get\_WorkBegin().get\_Hour() != 0)

listView1->Items[i]->SubItems->Add("");

else if (Main\_Days[Current\_Day - 1][i + 1].get\_Advent().get\_Minutes() < 10)

listView1->Items[i]->SubItems->Add(Main\_Days[Current\_Day - 1][i + 1].get\_Advent().get\_Hour().ToString() + ":" + "0" + Main\_Days[Current\_Day - 1][i + 1].get\_Advent().get\_Minutes().ToString());

else

listView1->Items[i]->SubItems->Add(Main\_Days[Current\_Day - 1][i + 1].get\_Advent().get\_Hour().ToString() + ":" + Main\_Days[Current\_Day - 1][i + 1].get\_Advent().get\_Minutes().ToString());

//Реальный уход

if (Main\_Days[Current\_Day - 1][i + 1].get\_Leave().get\_Hour() == 0 && Main\_Days[Current\_Day - 1][i + 1].get\_WorkEnd().get\_Hour() != 0)

listView1->Items[i]->SubItems->Add("");

else if (Main\_Days[Current\_Day - 1][i + 1].get\_Leave().get\_Minutes() < 10)

listView1->Items[i]->SubItems->Add(Main\_Days[Current\_Day - 1][i + 1].get\_Leave().get\_Hour().ToString() + ":" + "0" + Main\_Days[Current\_Day - 1][i + 1].get\_Leave().get\_Minutes().ToString());

else

listView1->Items[i]->SubItems->Add(Main\_Days[Current\_Day - 1][i + 1].get\_Leave().get\_Hour().ToString() + ":" + Main\_Days[Current\_Day - 1][i + 1].get\_Leave().get\_Minutes().ToString());

if (strcmp(Main\_Days[Current\_Day - 1][i + 1].get\_Attendance(), "П") == 0)

{

listView1->Items[i]->SubItems->Add("");

}

else if (strcmp(Main\_Days[Current\_Day - 1][i + 1].get\_Attendance(), "Н") == 0)

{

listView1->Items[i]->SubItems->Add("Нарушение");

}

else if (strcmp(Main\_Days[Current\_Day - 1][i + 1].get\_Attendance(), "У") == 0)

{

listView1->Items[i]->SubItems->Add("Ув. Причина");

}

}

}

}

}

System::Void Interface\_Main::button3\_Click(System::Object^ sender, System::EventArgs^ e)

{

Add\_Appearance^ AddAppearance = gcnew Add\_Appearance(this);

this->Hide();

AddAppearance->ShowDialog();

this->Show();

//Обновим данные о посещаемости на текущий день после прохода сотрудника:

//Очистим список от предыдущего списка рабочих:

for (int i = listView1->Items->Count - 1; i >= 0; i--)

{

listView1->Items[i]->Remove();

}

//Отобразим список рабочих текущего дня:

//Зададим цикл вывода списка для текущего(базового при загрузке) дня:

if (Main\_Days[Current\_Day - 1].size() != 1)

{

for (int i = 0; i < Main\_Days[Current\_Day - 1].size() - 1; i++)

{

if (Main\_Days[Current\_Day - 1][i + 1].get\_WorkBegin().get\_Hour() == 0 && Main\_Days[Current\_Day - 1][i + 1].get\_WorkBegin().get\_Minutes() == 0

&& Main\_Days[Current\_Day - 1][i + 1].get\_WorkEnd().get\_Hour() == 0 && Main\_Days[Current\_Day - 1][i + 1].get\_WorkEnd().get\_Minutes() == 0)

{

//Вывод для выходного дня:

listView1->Items->Add(Main\_Days[Current\_Day - 1][i + 1].get\_Number().ToString()); //Добавить поле Номер в первую колонку, далее последовательно привязываем другие поля экземпляра класса

listView1->Items[i]->SubItems->Add(gcnew System::String(Main\_Days[Current\_Day - 1][i + 1].get\_Name()));

listView1->Items[i]->SubItems->Add(gcnew System::String(Main\_Days[Current\_Day - 1][i + 1].get\_Surname()));

//Приход

listView1->Items[i]->SubItems->Add("-");

//Уход

listView1->Items[i]->SubItems->Add("-");

//Реальный приход

listView1->Items[i]->SubItems->Add("-");

//Реальный уход

listView1->Items[i]->SubItems->Add("-");

//Посещаемость

listView1->Items[i]->SubItems->Add("Выходной");

}

else

{

//Вывод для рабочего дня:

listView1->Items->Add(Main\_Days[Current\_Day - 1][i + 1].get\_Number().ToString()); //Добавить поле Номер в первую колонку, далее последовательно привязываем другие поля экземпляра класса

listView1->Items[i]->SubItems->Add(gcnew System::String(Main\_Days[Current\_Day - 1][i + 1].get\_Name()));

listView1->Items[i]->SubItems->Add(gcnew System::String(Main\_Days[Current\_Day - 1][i + 1].get\_Surname()));

//Приход

if (Main\_Days[Current\_Day - 1][i + 1].get\_WorkBegin().get\_Minutes() < 10)

listView1->Items[i]->SubItems->Add(Main\_Days[Current\_Day - 1][i + 1].get\_WorkBegin().get\_Hour().ToString() + ":" + "0" + Main\_Days[Current\_Day - 1][i + 1].get\_WorkBegin().get\_Minutes().ToString());

else

listView1->Items[i]->SubItems->Add(Main\_Days[Current\_Day - 1][i + 1].get\_WorkBegin().get\_Hour().ToString() + ":" + Main\_Days[Current\_Day - 1][i + 1].get\_WorkBegin().get\_Minutes().ToString());

//Уход

if (Main\_Days[Current\_Day - 1][i + 1].get\_WorkEnd().get\_Minutes() < 10)

listView1->Items[i]->SubItems->Add(Main\_Days[Current\_Day - 1][i + 1].get\_WorkEnd().get\_Hour().ToString() + ":" + "0" + Main\_Days[Current\_Day - 1][i + 1].get\_WorkEnd().get\_Minutes().ToString());

else

listView1->Items[i]->SubItems->Add(Main\_Days[Current\_Day - 1][i + 1].get\_WorkEnd().get\_Hour().ToString() + ":" + Main\_Days[Current\_Day - 1][i + 1].get\_WorkEnd().get\_Minutes().ToString());

//Реальный приход

if (Main\_Days[Current\_Day - 1][i + 1].get\_Advent().get\_Hour() == 0 && Main\_Days[Current\_Day - 1][i + 1].get\_WorkBegin().get\_Hour() != 0)

listView1->Items[i]->SubItems->Add("");

else if (Main\_Days[Current\_Day - 1][i + 1].get\_Advent().get\_Minutes() < 10)

listView1->Items[i]->SubItems->Add(Main\_Days[Current\_Day - 1][i + 1].get\_Advent().get\_Hour().ToString() + ":" + "0" + Main\_Days[Current\_Day - 1][i + 1].get\_Advent().get\_Minutes().ToString());

else

listView1->Items[i]->SubItems->Add(Main\_Days[Current\_Day - 1][i + 1].get\_Advent().get\_Hour().ToString() + ":" + Main\_Days[Current\_Day - 1][i + 1].get\_Advent().get\_Minutes().ToString());

//Реальный уход

if (Main\_Days[Current\_Day - 1][i + 1].get\_Leave().get\_Hour() == 0 && Main\_Days[Current\_Day - 1][i + 1].get\_WorkEnd().get\_Hour() != 0)

listView1->Items[i]->SubItems->Add("");

else if (Main\_Days[Current\_Day - 1][i + 1].get\_Leave().get\_Minutes() < 10)

listView1->Items[i]->SubItems->Add(Main\_Days[Current\_Day - 1][i + 1].get\_Leave().get\_Hour().ToString() + ":" + "0" + Main\_Days[Current\_Day - 1][i + 1].get\_Leave().get\_Minutes().ToString());

else

listView1->Items[i]->SubItems->Add(Main\_Days[Current\_Day - 1][i + 1].get\_Leave().get\_Hour().ToString() + ":" + Main\_Days[Current\_Day - 1][i + 1].get\_Leave().get\_Minutes().ToString());

if (strcmp(Main\_Days[Current\_Day - 1][i + 1].get\_Attendance(), "П") == 0)

{

listView1->Items[i]->SubItems->Add("");

}

else if (strcmp(Main\_Days[Current\_Day - 1][i + 1].get\_Attendance(), "Н") == 0)

{

listView1->Items[i]->SubItems->Add("Нарушение");

}

else if (strcmp(Main\_Days[Current\_Day - 1][i + 1].get\_Attendance(), "У") == 0)

{

listView1->Items[i]->SubItems->Add("Ув. Причина");

}

}

}

}

}

System::Void Interface\_Main::button4\_Click(System::Object^ sender, System::EventArgs^ e)

{

Violation\_Manager^ ViolationManager = gcnew Violation\_Manager(this);

this->Hide();

ViolationManager->ShowDialog();

this->Show();

//Обновим таблицу после возможной процедуры увольнения сотрудника:

//Очистим список от предыдущего списка рабочих:

for (int i = listView1->Items->Count - 1; i >= 0; i--)

{

listView1->Items[i]->Remove();

}

//Отобразим список рабочих текущего дня:

//Зададим цикл вывода списка для текущего(базового при загрузке) дня:

if (Main\_Days[Current\_Day - 1].size() != 1)

{

for (int i = 0; i < Main\_Days[Current\_Day - 1].size() - 1; i++)

{

if (Main\_Days[Current\_Day - 1][i + 1].get\_WorkBegin().get\_Hour() == 0 && Main\_Days[Current\_Day - 1][i + 1].get\_WorkBegin().get\_Minutes() == 0

&& Main\_Days[Current\_Day - 1][i + 1].get\_WorkEnd().get\_Hour() == 0 && Main\_Days[Current\_Day - 1][i + 1].get\_WorkEnd().get\_Minutes() == 0)

{

//Вывод для выходного дня:

listView1->Items->Add(Main\_Days[Current\_Day - 1][i + 1].get\_Number().ToString()); //Добавить поле Номер в первую колонку, далее последовательно привязываем другие поля экземпляра класса

listView1->Items[i]->SubItems->Add(gcnew System::String(Main\_Days[Current\_Day - 1][i + 1].get\_Name()));

listView1->Items[i]->SubItems->Add(gcnew System::String(Main\_Days[Current\_Day - 1][i + 1].get\_Surname()));

//Приход

listView1->Items[i]->SubItems->Add("-");

//Уход

listView1->Items[i]->SubItems->Add("-");

//Реальный приход

listView1->Items[i]->SubItems->Add("-");

//Реальный уход

listView1->Items[i]->SubItems->Add("-");

//Посещаемость

listView1->Items[i]->SubItems->Add("Выходной");

}

else

{

//Вывод для рабочего дня:

listView1->Items->Add(Main\_Days[Current\_Day - 1][i + 1].get\_Number().ToString()); //Добавить поле Номер в первую колонку, далее последовательно привязываем другие поля экземпляра класса

listView1->Items[i]->SubItems->Add(gcnew System::String(Main\_Days[Current\_Day - 1][i + 1].get\_Name()));

listView1->Items[i]->SubItems->Add(gcnew System::String(Main\_Days[Current\_Day - 1][i + 1].get\_Surname()));

//Приход

if (Main\_Days[Current\_Day - 1][i + 1].get\_WorkBegin().get\_Minutes() < 10)

listView1->Items[i]->SubItems->Add(Main\_Days[Current\_Day - 1][i + 1].get\_WorkBegin().get\_Hour().ToString() + ":" + "0" + Main\_Days[Current\_Day - 1][i + 1].get\_WorkBegin().get\_Minutes().ToString());

else

listView1->Items[i]->SubItems->Add(Main\_Days[Current\_Day - 1][i + 1].get\_WorkBegin().get\_Hour().ToString() + ":" + Main\_Days[Current\_Day - 1][i + 1].get\_WorkBegin().get\_Minutes().ToString());

//Уход

if (Main\_Days[Current\_Day - 1][i + 1].get\_WorkEnd().get\_Minutes() < 10)

listView1->Items[i]->SubItems->Add(Main\_Days[Current\_Day - 1][i + 1].get\_WorkEnd().get\_Hour().ToString() + ":" + "0" + Main\_Days[Current\_Day - 1][i + 1].get\_WorkEnd().get\_Minutes().ToString());

else

listView1->Items[i]->SubItems->Add(Main\_Days[Current\_Day - 1][i + 1].get\_WorkEnd().get\_Hour().ToString() + ":" + Main\_Days[Current\_Day - 1][i + 1].get\_WorkEnd().get\_Minutes().ToString());

//Реальный приход

if (Main\_Days[Current\_Day - 1][i + 1].get\_Advent().get\_Hour() == 0 && Main\_Days[Current\_Day - 1][i + 1].get\_WorkBegin().get\_Hour() != 0)

listView1->Items[i]->SubItems->Add("");

else if (Main\_Days[Current\_Day - 1][i + 1].get\_Advent().get\_Minutes() < 10)

listView1->Items[i]->SubItems->Add(Main\_Days[Current\_Day - 1][i + 1].get\_Advent().get\_Hour().ToString() + ":" + "0" + Main\_Days[Current\_Day - 1][i + 1].get\_Advent().get\_Minutes().ToString());

else

listView1->Items[i]->SubItems->Add(Main\_Days[Current\_Day - 1][i + 1].get\_Advent().get\_Hour().ToString() + ":" + Main\_Days[Current\_Day - 1][i + 1].get\_Advent().get\_Minutes().ToString());

//Реальный уход

if (Main\_Days[Current\_Day - 1][i + 1].get\_Leave().get\_Hour() == 0 && Main\_Days[Current\_Day - 1][i + 1].get\_WorkEnd().get\_Hour() != 0)

listView1->Items[i]->SubItems->Add("");

else if (Main\_Days[Current\_Day - 1][i + 1].get\_Leave().get\_Minutes() < 10)

listView1->Items[i]->SubItems->Add(Main\_Days[Current\_Day - 1][i + 1].get\_Leave().get\_Hour().ToString() + ":" + "0" + Main\_Days[Current\_Day - 1][i + 1].get\_Leave().get\_Minutes().ToString());

else

listView1->Items[i]->SubItems->Add(Main\_Days[Current\_Day - 1][i + 1].get\_Leave().get\_Hour().ToString() + ":" + Main\_Days[Current\_Day - 1][i + 1].get\_Leave().get\_Minutes().ToString());

if (strcmp(Main\_Days[Current\_Day - 1][i + 1].get\_Attendance(), "П") == 0)

{

listView1->Items[i]->SubItems->Add("");

}

else if (strcmp(Main\_Days[Current\_Day - 1][i + 1].get\_Attendance(), "Н") == 0)

{

listView1->Items[i]->SubItems->Add("Нарушение");

}

else if (strcmp(Main\_Days[Current\_Day - 1][i + 1].get\_Attendance(), "У") == 0)

{

listView1->Items[i]->SubItems->Add("Ув. Причина");

}

}

}

}

}

System::Void Interface\_Main::monthCalendar1\_DateSelected(System::Object^ sender, System::Windows::Forms::DateRangeEventArgs^ e)

{

//Сделаем смену списка для каждого из дней в году в момент выбора даты на календаре:

Changed\_Day = monthCalendar1->SelectionStart.DayOfYear;

//Очистим список от предыдущего списка рабочих:

for (int i = listView1->Items->Count - 1; i >= 0; i--)

{

listView1->Items[i]->Remove();

}

//Проставим новые отметки:

Main\_Days = Add\_Marks(Main\_Days, Current\_Day);

//Выведем текущий список рабочих:

if (Main\_Days[Changed\_Day - 1].size() != 1)

{

for (int i = 0; i < Main\_Days[Changed\_Day - 1].size() - 1; i++)

{

if (Main\_Days[Changed\_Day - 1][i + 1].get\_WorkBegin().get\_Hour() == 0 && Main\_Days[Changed\_Day - 1][i + 1].get\_WorkBegin().get\_Minutes() == 0

&& Main\_Days[Changed\_Day - 1][i + 1].get\_WorkEnd().get\_Hour() == 0 && Main\_Days[Changed\_Day - 1][i + 1].get\_WorkEnd().get\_Minutes() == 0)

{

//Вывод для выходного дня:

listView1->Items->Add(Main\_Days[Changed\_Day - 1][i + 1].get\_Number().ToString()); //Добавить поле Номер в первую колонку, далее последовательно привязываем другие поля экземпляра класса

listView1->Items[i]->SubItems->Add(gcnew System::String(Main\_Days[Changed\_Day - 1][i + 1].get\_Name()));

listView1->Items[i]->SubItems->Add(gcnew System::String(Main\_Days[Changed\_Day - 1][i + 1].get\_Surname()));

//Приход

listView1->Items[i]->SubItems->Add("-");

//Уход

listView1->Items[i]->SubItems->Add("-");

//Реальный приход

listView1->Items[i]->SubItems->Add("-");

//Реальный уход

listView1->Items[i]->SubItems->Add("-");

//Посещаемость

listView1->Items[i]->SubItems->Add("Выходной");

}

else

{

//Вывод для рабочего дня:

listView1->Items->Add(Main\_Days[Changed\_Day - 1][i + 1].get\_Number().ToString()); //Добавить поле Номер в первую колонку, далее последовательно привязываем другие поля экземпляра класса

listView1->Items[i]->SubItems->Add(gcnew System::String(Main\_Days[Changed\_Day - 1][i + 1].get\_Name()));

listView1->Items[i]->SubItems->Add(gcnew System::String(Main\_Days[Changed\_Day - 1][i + 1].get\_Surname()));

//Приход

if (Main\_Days[Changed\_Day - 1][i + 1].get\_WorkBegin().get\_Minutes() < 10)

listView1->Items[i]->SubItems->Add(Main\_Days[Changed\_Day - 1][i + 1].get\_WorkBegin().get\_Hour().ToString() + ":" + "0" + Main\_Days[Changed\_Day - 1][i + 1].get\_WorkBegin().get\_Minutes().ToString());

else

listView1->Items[i]->SubItems->Add(Main\_Days[Changed\_Day - 1][i + 1].get\_WorkBegin().get\_Hour().ToString() + ":" + Main\_Days[Changed\_Day - 1][i + 1].get\_WorkBegin().get\_Minutes().ToString());

//Уход

if (Main\_Days[Changed\_Day - 1][i + 1].get\_WorkEnd().get\_Minutes() < 10)

listView1->Items[i]->SubItems->Add(Main\_Days[Changed\_Day - 1][i + 1].get\_WorkEnd().get\_Hour().ToString() + ":" + "0" + Main\_Days[Changed\_Day - 1][i + 1].get\_WorkEnd().get\_Minutes().ToString());

else

listView1->Items[i]->SubItems->Add(Main\_Days[Changed\_Day - 1][i + 1].get\_WorkEnd().get\_Hour().ToString() + ":" + Main\_Days[Changed\_Day - 1][i + 1].get\_WorkEnd().get\_Minutes().ToString());

//Реальный приход

if (Main\_Days[Changed\_Day - 1][i + 1].get\_Advent().get\_Hour() == 0 && Main\_Days[Changed\_Day - 1][i + 1].get\_WorkBegin().get\_Hour() != 0)

listView1->Items[i]->SubItems->Add("");

else if (Main\_Days[Changed\_Day - 1][i + 1].get\_Advent().get\_Minutes() < 10)

listView1->Items[i]->SubItems->Add(Main\_Days[Changed\_Day - 1][i + 1].get\_Advent().get\_Hour().ToString() + ":" + "0" + Main\_Days[Changed\_Day - 1][i + 1].get\_Advent().get\_Minutes().ToString());

else

listView1->Items[i]->SubItems->Add(Main\_Days[Changed\_Day - 1][i + 1].get\_Advent().get\_Hour().ToString() + ":" + Main\_Days[Changed\_Day - 1][i + 1].get\_Advent().get\_Minutes().ToString());

//Реальный уход

if (Main\_Days[Changed\_Day - 1][i + 1].get\_Leave().get\_Hour() == 0 && Main\_Days[Changed\_Day - 1][i + 1].get\_WorkEnd().get\_Hour() != 0)

listView1->Items[i]->SubItems->Add("");

else if (Main\_Days[Changed\_Day - 1][i + 1].get\_Leave().get\_Minutes() < 10)

listView1->Items[i]->SubItems->Add(Main\_Days[Changed\_Day - 1][i + 1].get\_Leave().get\_Hour().ToString() + ":" + "0" + Main\_Days[Changed\_Day - 1][i + 1].get\_Leave().get\_Minutes().ToString());

else

listView1->Items[i]->SubItems->Add(Main\_Days[Changed\_Day - 1][i + 1].get\_Leave().get\_Hour().ToString() + ":" + Main\_Days[Changed\_Day - 1][i + 1].get\_Leave().get\_Minutes().ToString());

if (strcmp(Main\_Days[Changed\_Day - 1][i + 1].get\_Attendance(), "П") == 0)

{

listView1->Items[i]->SubItems->Add("");

}

else if (strcmp(Main\_Days[Changed\_Day - 1][i + 1].get\_Attendance(), "Н") == 0)

{

listView1->Items[i]->SubItems->Add("Нарушение");

}

else if (strcmp(Main\_Days[Changed\_Day - 1][i + 1].get\_Attendance(), "У") == 0)

{

listView1->Items[i]->SubItems->Add("Ув. Причина");

}

}

}

}

}

}

## Приложение 4. Форма добавления сотрудника Interface\_Add:

Interface\_Add.cpp:

#include "Interface\_Add.h"

namespace AMSystem

{

System::Void Interface\_Add::button1\_Click(System::Object^ sender, System::EventArgs^ e)

{

//В 2018 году 1 января - ПН, следовательно дни недели имеют порядковые номера от 1 до 7 января и так далее

if ((textBox1->Text != "") && (textBox2->Text != "") //Условие введенных фамилий

&&

//1-е условие как мининимум одного выбранного рабочего дня

!(comboBox1->Text == "" && comboBox2->Text == "" && comboBox3->Text == "" && comboBox4->Text == "" && comboBox5->Text == "" &&

comboBox6->Text == "" && comboBox7->Text == "" && comboBox8->Text == "" && comboBox9->Text == "" && comboBox10->Text == "" &&

comboBox11->Text == "" && comboBox12->Text == "" && comboBox13->Text == "" && comboBox14->Text == "")

&&

//2-е условие как минимум одного выбранного рабочего дня

!((comboBox1->Text != "" && comboBox2->Text == "") || (comboBox1->Text == "" && comboBox2->Text != "") ||

(comboBox3->Text != "" && comboBox4->Text == "") || (comboBox3->Text == "" && comboBox4->Text != "") ||

(comboBox5->Text != "" && comboBox6->Text == "") || (comboBox5->Text == "" && comboBox6->Text != "") ||

(comboBox7->Text != "" && comboBox8->Text == "") || (comboBox7->Text == "" && comboBox8->Text != "") ||

(comboBox9->Text != "" && comboBox10->Text == "") || (comboBox9->Text == "" && comboBox10->Text != "") ||

(comboBox11->Text != "" && comboBox12->Text == "") || (comboBox11->Text == "" && comboBox12->Text != "") ||

(comboBox13->Text != "" && comboBox14->Text == "") || (comboBox13->Text == "" && comboBox14->Text != "")

))

{

//Последовательно для каждого дня недели вызовем нашу функцию добавления работника:

//ПН:

Main\_Days = Add\_Stuff\_Member(Main\_Days, textBox1->Text, textBox2->Text, comboBox1->Text, comboBox2->Text, 1, Current\_Day);

//ВТ:

Main\_Days = Add\_Stuff\_Member(Main\_Days, textBox1->Text, textBox2->Text, comboBox3->Text, comboBox4->Text, 2, Current\_Day);

//СР:

Main\_Days = Add\_Stuff\_Member(Main\_Days, textBox1->Text, textBox2->Text, comboBox5->Text, comboBox6->Text, 3, Current\_Day);

//ЧТ:

Main\_Days = Add\_Stuff\_Member(Main\_Days, textBox1->Text, textBox2->Text, comboBox7->Text, comboBox8->Text, 4, Current\_Day);

//ПТ:

Main\_Days = Add\_Stuff\_Member(Main\_Days, textBox1->Text, textBox2->Text, comboBox9->Text, comboBox10->Text, 5, Current\_Day);

//СБ:

Main\_Days = Add\_Stuff\_Member(Main\_Days, textBox1->Text, textBox2->Text, comboBox11->Text, comboBox12->Text, 6, Current\_Day);

//ВС:

Main\_Days = Add\_Stuff\_Member(Main\_Days, textBox1->Text, textBox2->Text, comboBox13->Text, comboBox14->Text, 7, Current\_Day);

//Очистим поля ввода:

textBox1->Text = "";

textBox2->Text = "";

MessageBox::Show("Сотрудник был успешно зарегистрирован!", "", MessageBoxButtons::OK, MessageBoxIcon::Asterisk);

}

else

{

MessageBox::Show("Для добавление сотрудника необходимо обязательно указать его имя и фамилию, а так же график работы в рабочие дни, которых должно быть как минимум один!", "Ошибка ввода!", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

}

}

## Приложение 6. Форма регистрации прохода Add\_Appearance:

Add\_Appearance.cpp:

#include "Add\_Appearance.h"

using namespace System;

using namespace msclr::interop;

namespace AMSystem

{

//Внесем имена существующих на текущий день сотрудников в комбобокс:

System::Void Add\_Appearance::Add\_Appearance\_Load(System::Object^ sender, System::EventArgs^ e)

{

for (int i = 1; i < Main\_Days[Current\_Day - 1].size(); i++)

{

std::string s1 = Main\_Days[Current\_Day - 1][i].get\_Name();

std::string s2 = Main\_Days[Current\_Day - 1][i].get\_Surname();

std::string s = ". " + s1 + " " + s2;

String^ s\_buf = marshal\_as<String^>(s);

this->comboBox1->Items->Add(Main\_Days[Current\_Day - 1][i].get\_Number() + s\_buf);

}

}

//Событие изменения флажка посещения при выборе нового сотрудника:

System::Void Add\_Appearance::comboBox1\_SelectedIndexChanged(System::Object^ sender, System::EventArgs^ e)

{

flag = false;

}

//Регистрация прохода

System::Void Add\_Appearance::button1\_Click(System::Object^ sender, System::EventArgs^ e)

{

if (comboBox1->Text != "")

{

//Выделим уникальный номер сотрудника выбранного в комбобокс

char\* CurrentNumberChar = (char\*)(void\*)Marshal::StringToHGlobalAnsi(comboBox1->Text);

unsigned int CurrentNumber = 0;

CurrentNumberChar = (strtok(CurrentNumberChar, "."));

CurrentNumber = atoi(CurrentNumberChar);

//Выделим текущее время:

time\_t t;

struct tm \*t\_m;

t = time(NULL);

t\_m = localtime(&t);

unsigned int Hour, Minutes;

Hour = t\_m->tm\_hour;

Minutes = t\_m->tm\_min;

//Поместим время прохода соблюдая логические условия

if (strcmp(Main\_Days[Current\_Day - 1][CurrentNumber].get\_Attendance(), "В") == 0)

{

MessageBox::Show("Незачем проходить на территорию предприятия в ваш законный выходной день!", "", MessageBoxButtons::OK, MessageBoxIcon::Asterisk);

}

//На случай, если работник вошел, а программа была закрыта с сохранением данных:

else if ((Main\_Days[Current\_Day - 1][CurrentNumber].get\_Advent().get\_Hour() + Main\_Days[Current\_Day - 1][CurrentNumber].get\_Advent().get\_Minutes()) != 0 &&

(Main\_Days[Current\_Day - 1][CurrentNumber].get\_Leave().get\_Hour() == 0 && Main\_Days[Current\_Day - 1][CurrentNumber].get\_Leave().get\_Minutes() == 0) &&

(flag == false))

{

MessageBox::Show("Невозможно повторно прийти на работу!", "Ошибка ввода!", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

//Обработаем первый вход за день:

else if (Main\_Days[Current\_Day - 1][CurrentNumber].get\_Advent().get\_Hour() == 0 && Main\_Days[Current\_Day - 1][CurrentNumber].get\_Advent().get\_Minutes() == 0 &&

Main\_Days[Current\_Day - 1][CurrentNumber].get\_Leave().get\_Hour() == 0 && Main\_Days[Current\_Day - 1][CurrentNumber].get\_Leave().get\_Minutes() == 0 &&

flag == false)

{

flag = true;

Main\_Days[Current\_Day - 1][CurrentNumber].set\_Advent(Hour, Minutes);

String^ Date = gcnew String(DateFromDayNumber(Current\_Day - 1, Default\_Year, MonthDays, MonthDaysLeap).c\_str());

if (t\_m->tm\_min < 10)

{

MessageBox::Show("Вы прошли на территорию предприятия в " + t\_m->tm\_hour + ":" + "0" + t\_m->tm\_min + "." + " Дата прохода: " + Date, "", MessageBoxButtons::OK, MessageBoxIcon::Asterisk);

}

else

MessageBox::Show("Вы прошли на территорию предприятия в " + t\_m->tm\_hour + ":" + t\_m->tm\_min + "." + " Дата прохода: " + Date, "", MessageBoxButtons::OK, MessageBoxIcon::Asterisk);

}

//Обработаем дальнейшие входы, с выводом, но без записи их в память:

else if (flag == false)

{

flag = true;

String^ Date = gcnew String(DateFromDayNumber(Current\_Day - 1, Default\_Year, MonthDays, MonthDaysLeap).c\_str());

if (t\_m->tm\_min < 10)

{

MessageBox::Show("Вы прошли на территорию предприятия в " + t\_m->tm\_hour + ":" + "0" + t\_m->tm\_min + "." + " Дата прохода: " + Date, "", MessageBoxButtons::OK, MessageBoxIcon::Asterisk);

}

else

MessageBox::Show("Вы прошли на территорию предприятия в " + t\_m->tm\_hour + ":" + t\_m->tm\_min + "." + " Дата прохода: " + Date, "", MessageBoxButtons::OK, MessageBoxIcon::Asterisk);

}

else

{

MessageBox::Show("Невозможно повторно прийти на работу!", "Ошибка ввода!", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

}

else

{

MessageBox::Show("Вы не выбрали сотрудника!", "Ошибка ввода!", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

}

//Регистрация выхода

System::Void Add\_Appearance::button2\_Click(System::Object^ sender, System::EventArgs^ e)

{

if (comboBox1->Text != "")

{

//Выделим уникальный номер сотрудника выбранного в комбобокс

char\* CurrentNumberChar = (char\*)(void\*)Marshal::StringToHGlobalAnsi(comboBox1->Text);

unsigned int CurrentNumber = 0;

CurrentNumberChar = (strtok(CurrentNumberChar, "."));

CurrentNumber = atoi(CurrentNumberChar);

//Выделим текущее время:

time\_t t;

struct tm \*t\_m;

t = time(NULL);

t\_m = localtime(&t);

unsigned int Hour, Minutes;

Hour = t\_m->tm\_hour;

Minutes = t\_m->tm\_min;

//Поместим время прохода соблюдая логические условия

if (flag == true)

{

flag = false;

Main\_Days[Current\_Day - 1][CurrentNumber].set\_Leave(Hour, Minutes);

String^ Date = gcnew String(DateFromDayNumber(Current\_Day - 1, Default\_Year, MonthDays, MonthDaysLeap).c\_str());

if (t\_m->tm\_min < 10)

MessageBox::Show("Вы покинули территорию предприятия в " + t\_m->tm\_hour + ":" + "0" + t\_m->tm\_min + "." + " Дата выхода: " + Date, "", MessageBoxButtons::OK, MessageBoxIcon::Asterisk);

else

MessageBox::Show("Вы покинули территорию предприятия в " + t\_m->tm\_hour + ":" + t\_m->tm\_min + "." + " Дата выхода: " + Date, "", MessageBoxButtons::OK, MessageBoxIcon::Asterisk);

}

//На случай, если работник вошел, а программа была закрыта с сохранением данных:

else if ((Main\_Days[Current\_Day - 1][CurrentNumber].get\_Advent().get\_Hour() && Main\_Days[Current\_Day - 1][CurrentNumber].get\_Advent().get\_Minutes()) != 0 &&

(Main\_Days[Current\_Day - 1][CurrentNumber].get\_Leave().get\_Hour() == 0 && Main\_Days[Current\_Day - 1][CurrentNumber].get\_Leave().get\_Minutes() == 0) &&

(flag == false))

{

Main\_Days[Current\_Day - 1][CurrentNumber].set\_Leave(Hour, Minutes);

String^ Date = gcnew String(DateFromDayNumber(Current\_Day - 1, Default\_Year, MonthDays, MonthDaysLeap).c\_str());

if (t\_m->tm\_min < 10)

MessageBox::Show("Вы покинули территорию предприятия в " + t\_m->tm\_hour + ":" + "0" + t\_m->tm\_min + "." + " Дата выхода: " + Date, "", MessageBoxButtons::OK, MessageBoxIcon::Asterisk);

else

MessageBox::Show("Вы покинули территорию предприятия в " + t\_m->tm\_hour + ":" + t\_m->tm\_min + "." + " Дата выхода: " + Date, "", MessageBoxButtons::OK, MessageBoxIcon::Asterisk);

}

else

{

MessageBox::Show("Вы не можете повторно уйти с работы!", "Ошибка ввода!", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

}

else

{

MessageBox::Show("Вы не выбрали сотрудника!", "Ошибка ввода!", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

}

//Сообщить о уважительной причине:

System::Void Add\_Appearance::button3\_Click(System::Object^ sender, System::EventArgs^ e)

{

if (comboBox1->Text != "")

{

//Выделим уникальный номер сотрудника выбранного в комбобокс

char\* CurrentNumberChar = (char\*)(void\*)Marshal::StringToHGlobalAnsi(comboBox1->Text);

unsigned int CurrentNumber = 0;

CurrentNumberChar = (strtok(CurrentNumberChar, "."));

CurrentNumber = atoi(CurrentNumberChar);

//Добавим уважительную причину отсутствия в отметку о посещении:

if (strcmp(Main\_Days[Current\_Day - 1][CurrentNumber].get\_Attendance(), "У") != 0)

{

Main\_Days[Current\_Day - 1][CurrentNumber].set\_Attendance("У");

MessageBox::Show("Уважительная причина будет учтена!", "", MessageBoxButtons::OK, MessageBoxIcon::Asterisk);

}

else

{

MessageBox::Show("Вы уже сообщали о уважительной причине для этого дня!", "", MessageBoxButtons::OK, MessageBoxIcon::Asterisk);

}

}

else

{

MessageBox::Show("Вы не выбрали сотрудника!", "Ошибка ввода!", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

}

}

## Приложение 7. Форма управления нарушениями Violation\_Manager:

Violation\_Manager.cpp:

#include "Violation\_Manager.h"

using namespace System;

using namespace msclr::interop;

namespace AMSystem

{

//Загрузить данные о существуюших сотрудниках в комбобокс

System::Void Violation\_Manager::Violation\_Manager\_Load(System::Object^ sender, System::EventArgs^ e)

{

for (int i = 1; i < Main\_Days[Current\_Day - 1].size(); i++)

{

std::string s1 = Main\_Days[Current\_Day - 1][i].get\_Name();

std::string s2 = Main\_Days[Current\_Day - 1][i].get\_Surname();

std::string s = ". " + s1 + " " + s2;

String^ s\_buf = marshal\_as<String^>(s);

this->comboBox1->Items->Add(Main\_Days[Current\_Day - 1][i].get\_Number() + s\_buf);

}

}

//Нарушения за текущий месяц:

System::Void Violation\_Manager::button1\_Click(System::Object^ sender, System::EventArgs^ e)

{

//Очистим listview:

for (int i = listView1->Items->Count - 1; i >= 0; i--)

{

listView1->Items[i]->Remove();

}

if (comboBox1->Text != "")

{

//Выделим уникальный номер сотрудника выбранного в комбобокс

char\* CurrentNumberChar = (char\*)(void\*)Marshal::StringToHGlobalAnsi(comboBox1->Text);

unsigned int CurrentNumber = 0;

CurrentNumberChar = (strtok(CurrentNumberChar, "."));

CurrentNumber = atoi(CurrentNumberChar);

//Узнаем номер текущего месяца

time\_t t;

struct tm \*t\_m;

t = time(NULL);

t\_m = localtime(&t);

int Current\_Month = t\_m->tm\_mon;

//Опеределим номер первого дня текущего месяца в году

int MonthFirstDay = 0;

for (int i = 0; i < Current\_Month; i++)

{

if (MonthFirstDay < Current\_Day - 1)

MonthFirstDay += MonthDays[i];

}

//Выведем необходимый набор данных в listview:

int j = 0; //Переменная для listviewitems

for (int i = MonthFirstDay; i < Current\_Day - 1; i++)

{

//Условие проверки наличия работника в штате на тот день

if (CurrentNumber < Main\_Days[i].size() && (strcmp(Main\_Days[i][CurrentNumber].get\_Attendance(), "Н") == 0))

{

//Вывод для рабочего дня:

std::string Date = DateFromDayNumber(i, Default\_Year, MonthDays, MonthDaysLeap);

listView1->Items->Add(gcnew System::String(Date.c\_str())); //Добавить дату нарушения

//Приход

if (Main\_Days[i][CurrentNumber].get\_WorkBegin().get\_Minutes() < 10)

listView1->Items[j]->SubItems->Add(Main\_Days[i][CurrentNumber].get\_WorkBegin().get\_Hour().ToString() + ":" + "0" + Main\_Days[i][CurrentNumber].get\_WorkBegin().get\_Minutes().ToString());

else

listView1->Items[j]->SubItems->Add(Main\_Days[i][CurrentNumber].get\_WorkBegin().get\_Hour().ToString() + ":" + Main\_Days[i][CurrentNumber].get\_WorkBegin().get\_Minutes().ToString());

//Уход

if (Main\_Days[i][CurrentNumber].get\_WorkEnd().get\_Minutes() < 10)

listView1->Items[j]->SubItems->Add(Main\_Days[i][CurrentNumber].get\_WorkEnd().get\_Hour().ToString() + ":" + "0" + Main\_Days[i][CurrentNumber].get\_WorkEnd().get\_Minutes().ToString());

else

listView1->Items[j]->SubItems->Add(Main\_Days[i][CurrentNumber].get\_WorkEnd().get\_Hour().ToString() + ":" + Main\_Days[i][CurrentNumber].get\_WorkEnd().get\_Minutes().ToString());

//Реальный приход

if (Main\_Days[i][CurrentNumber].get\_Advent().get\_Hour() == 0 && Main\_Days[i][CurrentNumber].get\_WorkBegin().get\_Hour() != 0)

listView1->Items[j]->SubItems->Add("");

else if (Main\_Days[i][CurrentNumber].get\_Advent().get\_Minutes() < 10)

listView1->Items[j]->SubItems->Add(Main\_Days[i][CurrentNumber].get\_Advent().get\_Hour().ToString() + ":" + "0" + Main\_Days[i][CurrentNumber].get\_Advent().get\_Minutes().ToString());

else

listView1->Items[j]->SubItems->Add(Main\_Days[i][CurrentNumber].get\_Advent().get\_Hour().ToString() + ":" + Main\_Days[i][CurrentNumber].get\_Advent().get\_Minutes().ToString());

//Реальный уход

if (Main\_Days[i][CurrentNumber].get\_Leave().get\_Hour() == 0 && Main\_Days[i][CurrentNumber].get\_WorkEnd().get\_Hour() != 0)

listView1->Items[j]->SubItems->Add("");

else if (Main\_Days[i][CurrentNumber].get\_Leave().get\_Minutes() < 10)

listView1->Items[j]->SubItems->Add(Main\_Days[i][CurrentNumber].get\_Leave().get\_Hour().ToString() + ":" + "0" + Main\_Days[i][CurrentNumber].get\_Leave().get\_Minutes().ToString());

else

listView1->Items[j]->SubItems->Add(Main\_Days[i][CurrentNumber].get\_Leave().get\_Hour().ToString() + ":" + Main\_Days[i][CurrentNumber].get\_Leave().get\_Minutes().ToString());

j++;

}

}

}

else

{

MessageBox::Show("В данный момент нет доступных сотрудников!", "", MessageBoxButtons::OK, MessageBoxIcon::Asterisk);

}

}

//Нарушения за прошедший месяц:

System::Void Violation\_Manager::button2\_Click(System::Object^ sender, System::EventArgs^ e)

{

//Очистим listview:

for (int i = listView1->Items->Count - 1; i >= 0; i--)

{

listView1->Items[i]->Remove();

}

if (comboBox1->Text != "")

{

//Выделим уникальный номер сотрудника выбранного в комбобокс

char\* CurrentNumberChar = (char\*)(void\*)Marshal::StringToHGlobalAnsi(comboBox1->Text);

unsigned int CurrentNumber = 0;

CurrentNumberChar = (strtok(CurrentNumberChar, "."));

CurrentNumber = atoi(CurrentNumberChar);

//Узнаем номер текущего месяца

time\_t t;

struct tm \*t\_m;

t = time(NULL);

t\_m = localtime(&t);

int Current\_Month = t\_m->tm\_mon;

//В рамках года у января нет предыдущего месяца:

if (t\_m->tm\_mon != 0)

{

//Опеределим номер первого дня предыдущего месяца в году

int MonthFirstDay = 0;

int MonthLastDay = 0;

for (int i = 0; i < Current\_Month - 1; i++)

{

if (MonthFirstDay < Current\_Day - 1)

MonthFirstDay += MonthDays[i];

}

//Определим номер последнего дня предыдщего месяца в году

MonthLastDay = MonthFirstDay + MonthDays[Current\_Month];

//Выведем необходимый набор данных в listview:

int j = 0; //Переменная для listviewitems

for (int i = MonthFirstDay; i < MonthLastDay; i++)

{

//Условие проверки наличия работника в штате на тот день

if (CurrentNumber < Main\_Days[i].size() && (strcmp(Main\_Days[i][CurrentNumber].get\_Attendance(), "Н") == 0))

{

//Вывод для рабочего дня:

std::string Date = DateFromDayNumberForLastMonth(i, Default\_Year, MonthDays, MonthDaysLeap);

listView1->Items->Add(gcnew System::String(Date.c\_str())); //Добавить дату нарушения

//Приход

if (Main\_Days[i][CurrentNumber].get\_WorkBegin().get\_Minutes() < 10)

listView1->Items[j]->SubItems->Add(Main\_Days[i][CurrentNumber].get\_WorkBegin().get\_Hour().ToString() + ":" + "0" + Main\_Days[i][CurrentNumber].get\_WorkBegin().get\_Minutes().ToString());

else

listView1->Items[j]->SubItems->Add(Main\_Days[i][CurrentNumber].get\_WorkBegin().get\_Hour().ToString() + ":" + Main\_Days[i][CurrentNumber].get\_WorkBegin().get\_Minutes().ToString());

//Уход

if (Main\_Days[i][CurrentNumber].get\_WorkEnd().get\_Minutes() < 10)

listView1->Items[j]->SubItems->Add(Main\_Days[i][CurrentNumber].get\_WorkEnd().get\_Hour().ToString() + ":" + "0" + Main\_Days[i][CurrentNumber].get\_WorkEnd().get\_Minutes().ToString());

else

listView1->Items[j]->SubItems->Add(Main\_Days[i][CurrentNumber].get\_WorkEnd().get\_Hour().ToString() + ":" + Main\_Days[i][CurrentNumber].get\_WorkEnd().get\_Minutes().ToString());

//Реальный приход

if (Main\_Days[i][CurrentNumber].get\_Advent().get\_Hour() == 0 && Main\_Days[i][CurrentNumber].get\_WorkBegin().get\_Hour() != 0)

listView1->Items[j]->SubItems->Add("");

else if (Main\_Days[i][CurrentNumber].get\_Advent().get\_Minutes() < 10)

listView1->Items[j]->SubItems->Add(Main\_Days[i][CurrentNumber].get\_Advent().get\_Hour().ToString() + ":" + "0" + Main\_Days[i][CurrentNumber].get\_Advent().get\_Minutes().ToString());

else

listView1->Items[j]->SubItems->Add(Main\_Days[i][CurrentNumber].get\_Advent().get\_Hour().ToString() + ":" + Main\_Days[i][CurrentNumber].get\_Advent().get\_Minutes().ToString());

//Реальный уход

if (Main\_Days[i][CurrentNumber].get\_Leave().get\_Hour() == 0 && Main\_Days[i][CurrentNumber].get\_WorkEnd().get\_Hour() != 0)

listView1->Items[j]->SubItems->Add("");

else if (Main\_Days[i][CurrentNumber].get\_Leave().get\_Minutes() < 10)

listView1->Items[j]->SubItems->Add(Main\_Days[i][CurrentNumber].get\_Leave().get\_Hour().ToString() + ":" + "0" + Main\_Days[i][CurrentNumber].get\_Leave().get\_Minutes().ToString());

else

listView1->Items[j]->SubItems->Add(Main\_Days[i][CurrentNumber].get\_Leave().get\_Hour().ToString() + ":" + Main\_Days[i][CurrentNumber].get\_Leave().get\_Minutes().ToString());

j++;

}

}

}

else

{

MessageBox::Show("В рамках одного года у января не существует предыдущего месяца!", "", MessageBoxButtons::OK, MessageBoxIcon::Asterisk);

}

}

else

{

MessageBox::Show("В данный момент нет доступных сотрудников!", "", MessageBoxButtons::OK, MessageBoxIcon::Asterisk);

}

}

//Увольнение сотрудника:

System::Void Violation\_Manager::button3\_Click(System::Object^ sender, System::EventArgs^ e)

{

if (comboBox1->Text != "")

{

//Выделим уникальный номер сотрудника выбранного в комбобокс

char\* CurrentNumberChar = (char\*)(void\*)Marshal::StringToHGlobalAnsi(comboBox1->Text);

unsigned int CurrentNumber = 0;

CurrentNumberChar = (strtok(CurrentNumberChar, "."));

CurrentNumber = atoi(CurrentNumberChar);

if (Main\_Days[Current\_Day - 1].size() == 1)

MessageBox::Show("В данный момент нет доступных для удаления сотрудников!", "", MessageBoxButtons::OK, MessageBoxIcon::Asterisk);

else

{

Main\_Days = Dismiss\_Stuff\_Member(Main\_Days, Current\_Day, CurrentNumber);

//Удалим старый список комбобокса:

comboBox1->Items->Clear();

//Обновим комбобокс после удаления сотрудника:

for (int i = 1; i < Main\_Days[Current\_Day - 1].size(); i++)

{

std::string s1 = Main\_Days[Current\_Day - 1][i].get\_Name();

std::string s2 = Main\_Days[Current\_Day - 1][i].get\_Surname();

std::string s = ". " + s1 + " " + s2;

String^ s\_buf = marshal\_as<String^>(s);

this->comboBox1->Items->Add(Main\_Days[Current\_Day - 1][i].get\_Number() + s\_buf);

}

//Очистим список нарушений после увольнения сотрудника:

for (int i = listView1->Items->Count - 1; i >= 0; i--)

{

listView1->Items[i]->Remove();

}

}

}

else

{

MessageBox::Show("В данный момент нет доступных сотрудников!", "", MessageBoxButtons::OK, MessageBoxIcon::Asterisk);

}

}

}