**1. Изучение предметной области**

**1.1 Заинтересованные лица и их требования**

Заинтересованные лица:

1. Водитель
2. Бухгалтер
3. Депо
4. Диспетчер
5. Кондуктор

Основные требования:

* 1. Безошибочный ввод данных.
	2. Программа работает без сбоев.
	3. Хороший и удобный интерфейс.
	4. Быстрое и качественное обслуживание

**1.2 Документ «Видение»**

**Заинтересованные лица:**

**Водитель:** хочет осторожно и точно по расписанию производить маршрут.

**Бухгалтер:** хочет безошибочно и вовремя производить оплату с рабочими, точно отсчитывать прибыль и деньги в амортизационный фонд, в фонд налогов, за электроэнергию. Вовремя фиксировать необходимые данные в соответствующих журналах.

**Депо:** хочет, в случае необходимости производить ремонт – амортизацию транспорта.

**Кондуктор:** хочет зафиксировать в начале каждого маршрута количество билетов, выданных диспетчером, а также подсчитать их количество в конце маршрута.

**Диспетчер:** хочет правильно создавать путевые листы (расписание маршрутов), вовремя отправлять (принимать) транспорт на линию (с линии) маршрута. Следить за соответствием расписания и движения транспортов и регистрировать все отправления и прибытия транспортов в журнале регистрации.

**Место системы.**

Систему можно использовать на такой государственной организации, как городской электротранспорт.

**Пользователи системы.**

В качестве непосредственных пользователей системы могут выступать: водитель, диспетчер, бухгалтер и кондуктор городского электротранспорта.

**Задачи и свойства системы.**

1. Обеспечивать удобный интерфейс пользователя.
2. Реагировать на ошибки ввода – вывода данных.
3. Искать данные в системе по запросу.

**1.3 График выполнения курсовой работы**

|  |  |
| --- | --- |
| **Период выполнения** | **Выполняемый этап курсовой работы** |
| 2 неделя | Изучение предметной области. |
| 3\4 неделя | Выполнение анализа системы. |
| 5\9 неделя | Окончание проектирования. |
| 10\13 неделя | Написания программного кода. |
| 13\14 неделя | Предоставление окончательной программы. |
| 14\15 неделя | Защита курсовой работы. |

**1.4 Диаграмма прецедентов и описание прецедентов**

Диаграмма прецедентов

**Описание прецедентов**

**Прецедент П1**: Подсчет прибыли.

Основной исполнитель: Бухгалтер.

Заинтересованные лица и их требования:

Кондуктор: Хочет в начале и в конце каждого рабочего дня знать количество билетов, выданных диспетчером, и соответственно оставшихся.

Бухгалтер: Хочет подсчитать полученную прибыль от продажи билетов и распределить ее на оплату налогов и зарплаты.

Предусловия: Бухгалтер должен быть идентифицирован и аутентифицирован.

Постусловия: Прибыль посчитана, и данные о подсчете занесены в систему.

Основной, успешный сценарий:

1).В начале рабочего дня бухгалтер фиксирует в «журнале учета»: дату, количество выдающихся билетов, в соответствии с маршрутом; выдает билеты диспетчеру, который передает кондуктору.

2).В конце рабочего дня кондуктор возвращает диспетчеру оставшееся количество билетов и сумму от проданных. Диспетчер передает их бухгалтеру, который заносит всю необходимую информацию (дата, №маршрута, количество оставшихся билетов, количество выданных билетов, сумма) в «журнал учета».

3).Бухгалтер учитывает образовавшуюся прибыль и высчитывает денежные финансы на оплату налогов и заработной платы. А также, фиксирует количество денежных финансов на налоги в «журнале налогов», а заработную плату в «журнале ЗП».

Альтернативный сценарий 1:

2.а). В конце рабочего дня кондуктор возвращает диспетчеру всё оставшееся количество билетов, но не всю сумму денег за проданные билеты.

1. Диспетчер передает все бухгалтеру. Бухгалтер заносит всю необходимую информацию в систему.

2. Система при подсчете обнаруживает ошибку недостачи, а также количество штрафа.

3. Бухгалтер вызывает кондуктора и информирует его о недостаче. Кондуктор платит необходимую сумму штрафа, по ранее определенной сумме. Бухгалтер заносит заплаченную сумму штрафа в «журнал штрафов».

4. Бухгалтер заносит в систему уже новые данные. Переход к п. 1).

**Прецедент П2:** Распределение транспорта по маршрутам.

Основной исполнитель: Диспетчер.

Заинтересованные лица и их требования:

Диспетчер: Хочет точно и правильно создать путевые листы для каждого маршрута. Хочет правильно распределить транспорт по маршрутам, в соответствии с количеством пассажиров на маршрутах, а также не допустить пересечение или столкновение транспортов. Провести регистрацию в соответственном журнале.

Водитель: Хочет иметь в наличии маршрут своего транспорта, путевой лист, расписание остановок, точное время передвижения и остановок.

Предусловия: Диспетчер должен быть идентифицирован и аутентифицирован.

Постусловия: Транспорт распределен по маршрутам.

Основной, успешный сценарий.

1).Диспетчер создает путевые листы в системе для всех необходимых маршрутов, в зависимости от количества мест, приходящихся на каждый транспорт, и количеством пассажиров на маршрутах. Все путевые листы хранятся в «журнале П\_лист’.

2).В начале рабочего дня диспетчер выдает каждому водителю его путевой лист. Диспетчер отправляет транспорт по маршруту. Регистрирует начало каждого маршрута в «журнале регистрации отправки и прибытия транспортов».

3).Диспетчер принимает транспорт, проверяет его, и не найдя повреждений отпускает водителя, регистрирует в «регистрации отправки и прибытия транспортов», а также сообщает системе о завершении определенного маршрута.

Альтернативный сценарий 1:

3.а) В конце рабочего дня транспорт не возвращается в Депо

1. Диспетчер фиксирует в системе отсутствие транспорта. Информирует необходимых лиц. Производятся поиски транспорта.

2. При прибытии с маршрута транспорта переход к п. 3).

Альтернативный сценарий 2:

3.б) Принимая и осматривая транспорт в конце рабочего дня, диспетчер находит незначительные повреждения.

1. Диспетчер сообщает о повреждениях водителю, в Депо, в систему в «журнал повреждений».

2. Транспорт снимают с линии маршрута, заменяя его другим и фиксируя смену в системе.

3. Депо вызывает мастера по ремонтным работам, регистрируя в системе оплату работнику за ремонт.

4. После ремонта транспорт возвращают на линию маршрута. Переход к п. 3).

**Прецедент П3:** Расчет с работниками.

Основной исполнитель: Бухгалтер.

Заинтересованные лица и их требования:

Бухгалтер. Хочет точно и быстро выделить средства для оплаты услуг работников.

Руководство. Хочет аккуратно и точно записать и хранить информацию о выделенных средствах в систему.

Предусловия: Бухгалтер идентифицирован и аутентифицирован (имеет доступ к определенным ресурсам). Программа загружена.

Постусловия: Расчет с работниками. Занесение и сохранение соответствующей информации в журналах системы.

Основной, успешный сценарий

1).В систему бухгалтером заноситься количество выданных билетов и количество оставшихся, за весь месяц.

2).Система считает прибыль от всех проданных билетов. Бухгалтер отсчитывает некоторую часть прибыли на оплату налогов, заносит в «журнал налогов».

3).Система вычисляет заработную плату работникам в соответствии с установленным коэффициентом k, который (от всей суммы прибыли) составляет:

– для водителя = 0,3

– для кондуктора = 0,15

– для диспетчера = 0,15

– для бухгалтера =0,2

5).Система выводит конечные данные по оплате и сохраняет их. Бухгалтер расплачивается с работниками, фиксируя все расчеты в «журнале ЗП».

Альтернативный сценарий 1:

1.а). В систему бухгалтер заносит не всю информацию.

1. Бухгалтер заносит только количество выданных билетов.

2. Система выдает сообщение: «НЕДОСТАТОЧНО ДАННЫХ ДЛЯ РАСЧЕТА!!!»

3. Бухгалтер вводит уже всю информацию заново. Переход к п. 1.

**Прецедент П4:** Расчет с поставщиком электроэнергии.

Основной исполнитель: Бухгалтер.

Заинтересованные лица и их требования:

Бухгалтер. Хочет точно и быстро выделить средства для оплаты услуг энергопоставщика.

Руководство. Хочет аккуратно и точно записать в систему, и хранить информацию о выделенных энергопоставщику средствах.

Предусловия: Бухгалтер идентифицирован и аутентифицирован (имеет доступ к определенным ресурсам). Программа загружена.

Постусловия: Произведен расчет за электроэнергию. Занесение и сохранение соответствующей информации в базе данных системы.

Основной, успешный сценарий:

1).Бухгалтер заносит все необходимые данные (количество маршрутов, пройденные ими расстояния) в систему.

2).Система в конце каждого дня производит подсчет затраченной электроэнергии на каждый маршрут, соответственно учитывая расстояния маршрутов. Бухгалтер сохраняет в системе все данные в конце каждого дня.

3).В конце каждой недели бухгалтер суммирует окончательный результат за оплату электроэнергии. Бухгалтер сохраняет в системе все данные в конце каждой недели.

4).Бухгалтер представляет полученный отчет поставщику электроэнергии. Поставщик сверяет со своими расчетами, и при совпадении принимает оплату. Бухгалтер фиксирует в системе все проведенные расчеты и уплаты в «журнале оплаты за электричество».

Альтернативный сценарий 1:

4).Не совпадение расчетов бухгалтера и поставщика.

1. Бухгалтер заново вносит свои данные в систему. Система выдает новые данные об оплате.

2. Бухгалтер представляет новый полученный отчет поставщику электроэнергии. Переход к п4).

**1.5 Составление концептуальных классов**

Список категорий концептуальных классов

|  |  |
| --- | --- |
| **Категории концептуальных классов** | **Пример** |
| Физические или материальные объекты | Трамвай, троллейбус |
| Спецификации, элементы проектных решений или описание объектов | Описание регистрации |
| Места | Остановки, Депо |
| Транзакции | Регистрация |
| Роль людей | Водитель, кондуктор, диспетчер, бухгалтер |
| Контейнеры других объектов | Трамвай, троллейбус |
| Содержимое контейнеров | Пассажиры, кондуктор |
| Организации | Служба авторизации платежей, налоговая служба, амортизационная служба. |
| События | Продажа билета, создание путевого листа. |
| Правила и политика | Правило возврата путевого листа |
| Записи различных деятелей | Различного вида журналы |

Описание концептуальных классов.

Бухгалтер – Accountant

Пассажир – Passenger

Водитель – Driver

Диспетчер – Dispatch

Кондуктор – Conductor

Депо – Depo

Служба авторизации платежей – Service payment

Амортизационный фонд – Repair fund

Билет – Ticket

Налог – Tax

Прибыль – Profit

Заработная плата – Salary

Трамвай – Tram

Троллейбус – Trolley-bus

Путевой лист – Plist

Продажа – Sale

Оплата – Payment

Маршрут – Itinerary

Расписание – Time\_table

Налоговая служба – Tax\_Service

Энергопоставщик – ElSupplier

Журнал регистрации транспорта – Journal transport register

Журнал путевых листов – Journal\_Plist

Журнал учета – Journal\_Ychet

Журнал ЗП (Заработной платы) – Journal\_ZP

Журнал налогов – Journal\_Tax

Журнал оплаты за электроэнергию – Journal\_Elect

Журнал штрафов – Journal\_sh

Журнал повреждений – Journal\_break

Ассоциации классов

|  |  |
| --- | --- |
| **Категория** | **Пример** |
| А является физической частью В | Троллейбус =вагон |
| А физически содержится в В | Маршрут =остановка |
| А логически содержится в В | Остановка =расписание остановок |
| А получает В | Пассажир =билет |
| А начисляет В | Бухгалтер =зарплата |
| А использует В | Водитель = расписание |
| А выдает В | Диспетчер =путевой лист |
| А получает В | Водитель =путевой лист |
| А принимает В | Кондуктор =оплату |

Диаграмма концептуальных классов

Атрибуты классов

|  |
| --- |
| Itinerary |
| nameIt-ry: textCol. Stop: intnameStop: texttime between Stop: doubletimeA: doubletimeB: double |

|  |
| --- |
| Salary |
| Summa: doubleCol sale ticket: doubleBonus: doubleTax: doubleProcent: doubleHoliday: double |

|  |
| --- |
| PList |
| NumberT-t: intItinerary: texttimeA: doubletimeB: doublesurnameDriver: textyear: doublemonth: double |

|  |
| --- |
| Accountant |
| name: FIOaddres: texttel: PhoneNumber |

|  |
| --- |
| Transport\_Register |
| Surname\_Dispatch: textNumberIt-ry: doubleNumber\_Tr-t: doubletimeA: doubleTimeB: double |

|  |
| --- |
| Transport |
| Tip: textNumber: intSer\_number: int |

**2. Проектирование системы**

**2.1 Описание операций и диаграмм взаимодействия**

**Прецедент:** Распределение транспорта по маршрутам.

**Описание операции ОП 1:**

|  |  |
| --- | --- |
| Операция | Transport\_Itinerary |
| Ссылки | Распределение транспорта по маршрутам и занесение данных в журнал регистрации |
| Предусловия | Бухгалтер идентифицирован и аутентифицирован. |
| Постусловия | Транспорт распределен. Данные занесены в журнал. |

**Прецедент:** Начисление заработной платы.

Описание операции ОП 2:

|  |  |
| --- | --- |
| Операция | Receive\_Profit |
| Ссылки | Подсчет прибыли. |
| Предусловия | Бухгалтер идентифицирован и аутентифицирован. |
| Постусловия | Прибыль подсчитана, данные занесены в систему. |

Описание операции ОП 3:

|  |  |
| --- | --- |
| Операция | Pay\_Salary |
| Ссылки | Выделение средств оплаты услуг работникам |
| Предусловия | Бухгалтер идентифицирован и аутентифицирован. |
| Постусловия | Средства выделены, данные записаны в журнале системы. |

**Прецедент:** Оплата за электроэнергию.

Описание операции ОП 4:

|  |  |
| --- | --- |
| Операция | Pay\_Supplier |
| Ссылки | Выделение средств оплаты услуг поставщика энергии. |
| Предусловия | Бухгалтер идентифицирован и аутентифицирован |
| Постусловия | Средства выделены, данные записаны в журнале системы |

**2.2 Программные классы**

|  |
| --- |
| Journal\_Plist |
| FIO\_driver: StringFIO\_cond: String№marsh: Bytedata: Byte№Plist: Byte |
| Plist (№marsh, data, №Plist, FIO\_driver, FIO\_cond) |

|  |
| --- |
| Journal\_Ychet |
| data: Bytecolvo\_t №1: Bytecolvo\_t №2: Byte№marsh: Bytesum: Byte |
| Beginwork\_day (data, colvo\_t №1, №marsh)Endwork\_day (data, colvo\_t №1, colvo\_t №2, sum, №marsh) |

|  |
| --- |
| Journal\_ZP |
| pribul: BytesumZP: Bytezp: Bytezp\_account: Bytezp\_driv: Bytezp\_disp: Bytezp\_cond: Byte |
| Podschet\_ZP (pribul, sumZP)Pay\_ZP (zp, zp\_account, zp\_driv, zp\_disp, zp\_cond) |

|  |
| --- |
| Journal\_transport register |
| data: Byte№marsh: BytetimeA: BytetimeB: Byte |
| Begin\_marsh (data, №marsh, timeA)End\_marsh (data, №marsh, timeB) |

|  |
| --- |
| Journal\_sh |
| №marsh: Bytesum\_sh: Bytedata: ByteFIO: String |
| Shtraff (sum\_sh, data, FIO, №marsh) |

|  |
| --- |
| Journal\_Tax |
| pribul: BytesumTax: Bytedata: Byte |
| Podschet\_Tax (pribul, sumTax)Pay\_ZP (sumTax, data) |

|  |
| --- |
| Journal\_break |
| №marsh: Bytedata: Byte |
| Polomka (data, №marsh) |

|  |
| --- |
| Journal\_Elect |
| data: Bytesum\_el: Byte |
| El\_oplata (data, sum\_el) |

|  |
| --- |
| System |
| FIO\_driver: StringFIO\_cond: String№marsh: Bytedata: Byte№Plist: Bytecolvo\_t №1: Bytecolvo\_t №2: Bytesum: Bytepribul: BytesumZP: Bytezp: Bytezp\_account: Bytezp\_driv: Bytezp\_disp: Bytezp\_cond: Bytedata: BytetimeA: BytetimeB: Bytesum\_sh: ByteFIO: Stringpribul: BytesumTax: Bytesum\_el: Bytetime\_now: Byte№marsh\_old: Byte№marsh\_new: Bytesum\_pay: Byteall\_prible: Byte |
| Plist (№marsh, data, №Plist, FIO\_driver, FIO\_cond), Beginwork\_day (data, colvo\_t №1, №marsh), Endwork\_day (data, colvo\_t №1, colvo\_t №2, sum, №marsh),Podschet\_ZP (pribul, sumZP), Pay\_ZP (zp, zp\_account, zp\_driv, zp\_disp, zp\_cond),Begin\_marsh (data, №marsh, timeA), End\_marsh (data, №marsh, timeB), Shtraff (sum\_sh, data, FIO, №marsh), Podschet\_Tax (pribul, sumTax), Pay\_ZP (sumTax, data), Polomka (data, №marsh), El\_oplata (data, sum\_el), Otsyts\_tr (FIO\_driver, FIO\_cond, data, time\_now, №marsh), Zamena (№marsh\_old, №marsh\_new), Pay\_break (sum\_pay, data), Salary (all\_prible, data) |

**3. Описание интерфейса приложения**

При входе в систему, она запрашивает пароль. Без него пользователь не сможет иметь доступ к системе.

Далее, пользователь может выбирать необходимое действие, нажав в открывшемся окне – File -> Action.

В результате, получив список, необходимых действий.

Для начала выбираем создание путевого листа (Path List). В необходимых колонках вводим соответствующие данные. После создания обязательно сохраняем в созданном и указанном журнале.

Сделаем проверку журнала, в котором должен был сохраниться наш 1-й путевой лист.

Далее выбираем учет билетов (Uchet). Здесь аналогично вводим необходимые данные. Также сохраняем в журнале.

Проверяем журнал.

Далее выбираем операции с заработной платой и налогом. Введя необходимые данные, сохраняем в журнале.

Проверяем выданное и оставшееся количество билетов.

Начисляем заработную плату.

Проверяем начисление зарплаты.

Выбрав пункт Поломка (Polomka), мы видим следующее окно, в котором вводим необходимые данные.

Проверяем в журнале.

Если необходимо произвести отчет, то нажав на главной панели кнопочку Browse, мы видим следующее окно.

В котором выбрав необходимый нам журнал или файл, видим его в окне на главной панели.

И дополнительная информация об авторе этого замечательного проекта может быть найдена по адресу – About -> Show.


##### Заключение

В результате выполнения данного курсового проекта было разработано приложение для автоматизации работы службы занятости. Данное приложение позволяет значительно упростить работу сотрудников службы занятости (инспектора и бухгалтера) и сэкономить время посетителей данной организации, которыми являются безработные и работодатели. Программа позволяет оформлять и хранить заявки посетителей, быстро находить вакантные места для безработных и рабочих для работодателей, содержит информацию о курсах переобучения безработных, позволяет начислять пособие безработным, заявки которых сохранены. Предусмотрены разные варианты обращения посетителей. Правила пользования программой описаны в разделе Описание интерфейса.

**Список литературы**

1. Майкл Дж. Янг Полное руководство по Visual C++ 6. Издательский дом «Ирина», BHV, Киев, 2000 г.-1048 с.

2. Дейтел Х.М., Дейтел П.Дж. Как программировать на С++ (Третье издание). Под издательством ЗАО «Бином», Москва, 2003 г.-1150 с.

3. Крэг Лерман. Применение UML и шаблонов проектирования. М. Издательский дом «Вильямс». 2002.-624 с.

4. Гради Буч. Объектно-ориентированный анализ и проектирование. Бином, СПБ, 2000. -560 с.

5. Орлов С.А. Технология разработки программного обеспечения. СПБ.: Питер, 2002. -464 с.

**Приложение**

**Листинги программных модулей**

**Add class**

#include «add1.h»

#include «path.h»

#include «uch.h»

#include «bre.h»

#include «elec.h»

void add: OnChoose()

{

UpdateData(TRUE);

if (m\_radio==0) {

path dlg;

dlg. DoModal();

}

if (m\_radio==1) {

uch dlg;

dlg. DoModal();

}

if (m\_radio==2) {

add1 dlg;

dlg. DoModal();

}

if (m\_radio==3) {

elec dlg;

dlg. DoModal();

}

if (m\_radio==4) {

bre dlg;

dlg. DoModal();

}

}

**Add1 class**

// add1.cpp: implementation file

 //

#include «stdafx.h»

#include «MasKurs.h»

#include «add1.h»

#ifdef \_DEBUG

#define new DEBUG\_NEW

#undef THIS\_FILE

static char THIS\_FILE[] = \_\_FILE\_\_;

#endif

 /////////////////////////////////////////////////////////////////////////////

 // add1 dialog

add1:add1 (CWnd\* pParent /\*=NULL\*/)

: CDialog (add1:IDD, pParent)

{

 // {{AFX\_DATA\_INIT(add1)

m\_date = \_T(«»);

m\_numberofout = \_T(«»);

m\_numberofleft = \_T(«»);

m\_number = \_T(«»);

m\_profit = 0;

m\_zarplata = 0;

m\_zarpl = \_T(«»);

m\_nalog1 = \_T(«»);

m\_profit1 = 0;

m\_nalog = 0;

m\_shtraf = \_T(«»);

 // }} AFX\_DATA\_INIT

}

void add1: DoDataExchange (CDataExchange\* pDX)

{

CDialog: DoDataExchange(pDX);

 // {{AFX\_DATA\_MAP(add1)

DDX\_Control (pDX, IDC\_COMBO1, m\_combo1);

DDX\_Text (pDX, IDC\_EDIT1, m\_date);

DDX\_Text (pDX, IDC\_EDIT2, m\_numberofout);

DDX\_Text (pDX, IDC\_EDIT3, m\_numberofleft);

DDX\_Text (pDX, IDC\_EDIT11, m\_number);

DDX\_Text (pDX, IDC\_EDIT4, m\_profit);

DDX\_Text (pDX, IDC\_EDIT5, m\_zarplata);

DDX\_Text (pDX, IDC\_EDIT8, m\_zarpl);

DDX\_Text (pDX, IDC\_EDIT9, m\_nalog1);

DDX\_Text (pDX, IDC\_EDIT6, m\_profit1);

DDX\_Text (pDX, IDC\_EDIT7, m\_nalog);

DDX\_Text (pDX, IDC\_EDIT10, m\_shtraf);

 // }} AFX\_DATA\_MAP

}

BEGIN\_MESSAGE\_MAP (add1, CDialog)

 // {{AFX\_MSG\_MAP(add1)

ON\_WM\_SHOWWINDOW()

ON\_BN\_CLICKED (IDC\_BUTTON5, OnSaveToUchetZarplataNalogi)

ON\_BN\_CLICKED (IDC\_BUTTON1, OnCount1)

ON\_BN\_CLICKED (IDC\_BUTTON2, OnSaveToZarp)

ON\_BN\_CLICKED (IDC\_BUTTON3, OnCount2)

ON\_BN\_CLICKED (IDC\_BUTTON4, OnSaveToNalog)

ON\_BN\_CLICKED (IDC\_BUTTON6, OnSaveToshtraf)

 // }} AFX\_MSG\_MAP

END\_MESSAGE\_MAP()

 /////////////////////////////////////////////////////////////////////////////

 // add1 message handlers

CString str0;

void add1: OnShowWindow (BOOL bShow, UINT nStatus)

{

CDialog: OnShowWindow (bShow, nStatus);

UpdateData(TRUE);

 // CStdioFile fOpen («d:\\MasKurs\\driver.txt», CFile:modeRead);

CStdioFile fOpen1 («d:\\MasKurs\\konductor.txt», CFile:modeRead);

CString str;

 // CString str1;

 // while (fOpen. ReadString(str)!=0) {

 // m\_combo. AddString(str);

 // }

while (fOpen1. ReadString(str0)!=0) {

m\_combo1. AddString(str0);

}

}

void add1: OnSaveToUchetZarplataNalogi()

{

UpdateData(TRUE);

CStdioFile fOpen («d:\\MasKurs\\uchetzarplatnalogi.txt», CFile:modeWrite);

CString str;

 // int index=m\_combo. GetCurSel();

int index1=m\_combo1. GetCurSel();

 // m\_combo. GetLBText (index, str1);

m\_combo1. GetLBText (index1, str0);

if (m\_date==»» || m\_number==»» || m\_numberofout==»» || m\_numberofleft==»» || str0==»») {

MessageBox («Please fill all fields», «Info», MB\_OK);

return;

}

fOpen. SeekToEnd();

 // fOpen. WriteString («FIO driver «+str1+»\n»);

fOpen. WriteString («FIO konductor «+str0+»\n»);

fOpen. WriteString («Date «+m\_date+»\n»);

fOpen. WriteString («Number of vudanih «+m\_numberofout+»\n»);

fOpen. WriteString («Number of ostavshihsa «+m\_numberofleft+»\n»);

fOpen. WriteString («Number «+m\_number+»\n»);

fOpen. WriteString («\*\*\*\*\*\n»);

MessageBox («Saved to uchetzarplatnalogi.txt», MB\_OK);

}

void add1: OnCount1 ()

{

UpdateData(TRUE);

int result=m\_profit\*m\_zarplata/100;

CString str;

str. Format («%d», result);

GetDlgItem (IDC\_EDIT8)->SetWindowText(str);

 // UpdateData(FALSE);

}

void add1: OnSaveToZarp()

{

UpdateData(TRUE);

CStdioFile fOpen («d:\\MasKurs\\zarplata.txt», CFile:modeWrite);

CString str;

 // int index=m\_combo. GetCurSel();

int index1=m\_combo1. GetCurSel();

 // m\_combo. GetLBText (index, str1);

m\_combo1. GetLBText (index1, str0);

if (m\_profit==0 || m\_zarplata==0 || str0==»») {

MessageBox («Please fill all fields», «Info», MB\_OK);

return;

}

fOpen. SeekToEnd();

 // fOpen. WriteString («FIO driver «+str1+»\n»);

fOpen. WriteString («FIO konductor «+str0+»\n»);

fOpen. WriteString («Zarplata «+m\_zarpl+»\n»);

fOpen. WriteString («\*\*\*\*\*\n»);

MessageBox («Saved to zarplata.txt», MB\_OK);

}

void add1: OnCount2 ()

{

UpdateData(TRUE);

int result=m\_profit1\*m\_nalog/100;

CString str;

str. Format («%d», result);

 // GetDlgItem (IDC\_EDIT9)->SetWindowText(str);

m\_nalog1=str;

UpdateData(FALSE);

}

void add1: OnSaveToNalog()

{

UpdateData(TRUE);

CStdioFile fOpen («d:\\MasKurs\\nalog.txt», CFile:modeWrite);

CString str;

 // int index=m\_combo. GetCurSel();

 // int index1=m\_combo1. GetCurSel();

 // m\_combo. GetLBText (index, str1);

 // m\_combo1. GetLBText (index1, str0);

if (m\_profit1==0 || m\_nalog==0) {

MessageBox («Please fill all fields», «Info», MB\_OK);

return;

}

fOpen. SeekToEnd();

str. Format («%d», m\_profit1);

 // fOpen. WriteString («FIO driver «+str1+»\n»);

fOpen. WriteString («Profit «+str+»\n»);

fOpen. WriteString («Nalog «+m\_nalog1+»\n»);

fOpen. WriteString («\*\*\*\*\*\n»);

MessageBox («Saved to nalog.txt», MB\_OK);

}

void add1: OnSaveToshtraf()

{

UpdateData(TRUE);

CStdioFile fOpen («d:\\MasKurs\\shtraf.txt», CFile:modeWrite);

CString str;

 // int index=m\_combo. GetCurSel();

int index1=m\_combo1. GetCurSel();

 // m\_combo. GetLBText (index, str1);

m\_combo1. GetLBText (index1, str0);

if (m\_shtraf==»» || str0==»») {

MessageBox («Please fill all fields», «Info», MB\_OK);

return;

}

fOpen. SeekToEnd();

 // fOpen. WriteString («FIO driver «+str1+»\n»);

fOpen. WriteString («FIO konductor «+str0+»\n»);

fOpen. WriteString («Shtraf «+m\_shtraf+»\n»);

fOpen. WriteString («\*\*\*\*\*\n»);

MessageBox («Saved to shtraf.txt», MB\_OK);

}

**Bre class**

void bre: OnSaveToBreak()

{

UpdateData(TRUE);

CStdioFile fOpen («d:\\MasKurs\\bre.txt», CFile:modeWrite);

CString str;

if (m\_date==»» || m\_number==»») {

MessageBox («Please fill all fields», «Info», MB\_OK);

return;

}

fOpen. SeekToEnd();

fOpen. WriteString («Date «+m\_date+»\n»);

fOpen. WriteString («Number «+m\_number+»\n»);

fOpen. WriteString («\*\*\*\*\*\n»);

MessageBox («Saved to bre.txt», MB\_OK);

}

void bre: OnSaveToZamena()

{

UpdateData(TRUE);

CStdioFile fOpen («d:\\MasKurs\\zamena.txt», CFile:modeWrite);

CString str;

if (m\_date1==»» || m\_number1==»» || m\_number2==»») {

MessageBox («Please fill all fields», «Info», MB\_OK);

return;

}

fOpen. SeekToEnd();

fOpen. WriteString («Number marsh old «+m\_number1+»\n»);

fOpen. WriteString («Number marsh new «+m\_number2+»\n»);

fOpen. WriteString («Date «+m\_date1+»\n»);

fOpen. WriteString («\*\*\*\*\*\n»);

MessageBox («Saved to zamena.txt», MB\_OK);

}

void bre: OnSaveToPay()

{

UpdateData(TRUE);

CStdioFile fOpen («d:\\MasKurs\\pay.txt», CFile:modeWrite);

CString str;

if (m\_date2==»» || m\_sum==»») {

MessageBox («Please fill all fields», «Info», MB\_OK);

return;

}

fOpen. SeekToEnd();

fOpen. WriteString («Sum «+m\_sum+»\n»);

fOpen. WriteString («Date «+m\_date2+»\n»);

fOpen. WriteString («\*\*\*\*\*\n»);

MessageBox («Saved to pay.txt», MB\_OK);

}

**Elec class**

void elec: OnSaveToElectro()

{

UpdateData(TRUE);

CStdioFile fOpen («d:\\MasKurs\\elec.txt», CFile:modeWrite);

CString str;

 // m\_combo. GetLBText (index, str1);

 // m\_combo1. GetLBText (index1, str0);

if (m\_ammount==»» || m\_sum==»») {

MessageBox («Please fill all fields», «Info», MB\_OK);

return;

}

fOpen. SeekToEnd();

 // fOpen. WriteString («FIO driver «+str1+»\n»);

fOpen. WriteString («Ammount «+m\_ammount+»\n»);

fOpen. WriteString («Number «+m\_sum+»\n»);

fOpen. WriteString («\*\*\*\*\*\n»);

MessageBox («Saved to elec.txt», MB\_OK);

}

**MasKursDlg class**

// MasKursDlg.cpp: implementation file

 //

#include «stdafx.h»

#include «MasKurs.h»

#include «MasKursDlg.h»

#ifdef \_DEBUG

#define new DEBUG\_NEW

#undef THIS\_FILE

static char THIS\_FILE[] = \_\_FILE\_\_;

#endif

 /////////////////////////////////////////////////////////////////////////////

 // CAboutDlg dialog used for App About

class CAboutDlg: public CDialog

{

public:

CAboutDlg();

 // Dialog Data

 // {{AFX\_DATA(CAboutDlg)

enum {IDD = IDD\_ABOUTBOX};

 // }} AFX\_DATA

 // ClassWizard generated virtual function overrides

 // {{AFX\_VIRTUAL(CAboutDlg)

protected:

virtual void DoDataExchange (CDataExchange\* pDX); // DDX/DDV support

 // }} AFX\_VIRTUAL

 // Implementation

protected:

 // {{AFX\_MSG(CAboutDlg)

 // }} AFX\_MSG

DECLARE\_MESSAGE\_MAP()

};

CAboutDlg:CAboutDlg(): CDialog (CAboutDlg:IDD)

{

 // {{AFX\_DATA\_INIT(CAboutDlg)

 // }} AFX\_DATA\_INIT

}

void CAboutDlg: DoDataExchange (CDataExchange\* pDX)

{

CDialog: DoDataExchange(pDX);

 // {{AFX\_DATA\_MAP(CAboutDlg)

 // }} AFX\_DATA\_MAP

}

BEGIN\_MESSAGE\_MAP (CAboutDlg, CDialog)

 // {{AFX\_MSG\_MAP(CAboutDlg)

 // No message handlers

 // }} AFX\_MSG\_MAP

END\_MESSAGE\_MAP()

 /////////////////////////////////////////////////////////////////////////////

 // CMasKursDlg dialog

CMasKursDlg:CMasKursDlg (CWnd\* pParent /\*=NULL\*/)

: CDialog (CMasKursDlg:IDD, pParent)

{

 // {{AFX\_DATA\_INIT(CMasKursDlg)

 // NOTE: the ClassWizard will add member initialization here

 // }} AFX\_DATA\_INIT

 // Note that LoadIcon does not require a subsequent DestroyIcon in Win32

m\_hIcon = AfxGetApp()->LoadIcon (IDR\_MAINFRAME);

}

void CMasKursDlg: DoDataExchange (CDataExchange\* pDX)

{

CDialog: DoDataExchange(pDX);

 // {{AFX\_DATA\_MAP(CMasKursDlg)

DDX\_Control (pDX, IDC\_LIST1, m\_list);

 // }} AFX\_DATA\_MAP

}

BEGIN\_MESSAGE\_MAP (CMasKursDlg, CDialog)

 // {{AFX\_MSG\_MAP(CMasKursDlg)

ON\_WM\_SYSCOMMAND()

ON\_WM\_PAINT()

ON\_WM\_QUERYDRAGICON()

ON\_COMMAND (ID\_FILE\_ACTION, OnFileAction)

ON\_BN\_CLICKED (IDC\_BUTTON3, OnBrowse)

ON\_BN\_CLICKED (IDC\_BUTTON1, OnButton1)

ON\_BN\_CLICKED (IDC\_BUTTON2, OnButton2)

ON\_COMMAND (ID\_FILE\_EXIT, OnFileExit)

ON\_COMMAND (ID\_ABOUT\_SHOW, OnAboutShow)

 // }} AFX\_MSG\_MAP

END\_MESSAGE\_MAP()

 /////////////////////////////////////////////////////////////////////////////

 // CMasKursDlg message handlers

#include «login.h»

BOOL CMasKursDlg: OnInitDialog()

{

CDialog: OnInitDialog();

 // Add «About…» menu item to system menu.

 // IDM\_ABOUTBOX must be in the system command range.

ASSERT((IDM\_ABOUTBOX & 0xFFF0) == IDM\_ABOUTBOX);

ASSERT (IDM\_ABOUTBOX < 0xF000);

CMenu\* pSysMenu = GetSystemMenu(FALSE);

if (pSysMenu!= NULL)

{

CString strAboutMenu;

strAboutMenu. LoadString (IDS\_ABOUTBOX);

if (! strAboutMenu. IsEmpty())

{

pSysMenu->AppendMenu (MF\_SEPARATOR);

pSysMenu->AppendMenu (MF\_STRING, IDM\_ABOUTBOX, strAboutMenu);

}

}

login dlg;

dlg.m\_login= «buhgalter»;

if (dlg. DoModal()==IDOK) {

if (dlg.m\_pass== «111» && dlg.m\_login== «buhgalter») {

return TRUE;

}

else {

MessageBox («Incorrect!!!»);

EndDialog(0);

}

}

 // Set the icon for this dialog. The framework does this automatically

 // when the application's main window is not a dialog

SetIcon (m\_hIcon, TRUE); // Set big icon

SetIcon (m\_hIcon, FALSE); // Set small icon

 // TODO: Add extra initialization here

return TRUE; // return TRUE unless you set the focus to a control

}

void CMasKursDlg: OnSysCommand (UINT nID, LPARAM lParam)

{

if ((nID & 0xFFF0) == IDM\_ABOUTBOX)

{

CAboutDlg dlgAbout;

dlgAbout. DoModal();

}

else

{

CDialog: OnSysCommand (nID, lParam);

}

}

 // If you add a minimize button to your dialog, you will need the code below

 // to draw the icon. For MFC applications using the document/view model,

 // this is automatically done for you by the framework.

void CMasKursDlg: OnPaint()

{

if (IsIconic())

{

CPaintDC dc(this); // device context for painting

SendMessage (WM\_ICONERASEBKGND, (WPARAM) dc. GetSafeHdc(), 0);

 // Center icon in client rectangle

int cxIcon = GetSystemMetrics (SM\_CXICON);

int cyIcon = GetSystemMetrics (SM\_CYICON);

CRect rect;

GetClientRect(&rect);

int x = (rect. Width() – cxIcon + 1) / 2;

int y = (rect. Height() – cyIcon + 1) / 2;

 // Draw the icon

dc. DrawIcon (x, y, m\_hIcon);

}

else

{

CDialog: OnPaint();

}

}

 // The system calls this to obtain the cursor to display while the user drags

 // the minimized window.

HCURSOR CMasKursDlg: OnQueryDragIcon()

{

return (HCURSOR) m\_hIcon;

}

#include «add.h»

void CMasKursDlg: OnFileAction()

{

add dlg;

dlg. DoModal();

}

void CMasKursDlg: OnBrowse()

{

CString str;

CString str1;

CFileDialog OpenFileDialog (true, «txt», NULL, OFN\_FILEMUSTEXIST|OFN\_HIDEREADONLY, «Text files (\*.txt)|\*.txt||»);

if (OpenFileDialog. DoModal()==IDOK) {

str=OpenFileDialog. GetPathName();

GetDlgItem (IDC\_EDIT1)->SetWindowText(str);

CStdioFile fOpen (str, CFile:modeRead);

while (fOpen. ReadString(str)!=0) {

m\_list. AddString(str);

}

}

}

void CMasKursDlg: OnButton1 ()

{

add dlg;

dlg. DoModal();

}

void CMasKursDlg: OnButton2 ()

{

EndDialog(0);

}

void CMasKursDlg: OnFileExit()

{

EndDialog(0);

}

void CMasKursDlg: OnAboutShow()

{

CAboutDlg dlg;

dlg. DoModal();

}

**Path class**

void path: OnShowWindow (BOOL bShow, UINT nStatus)

{

CDialog: OnShowWindow (bShow, nStatus);

UpdateData(TRUE);

CStdioFile fOpen («d:\\MasKurs\\driver.txt», CFile:modeRead);

CStdioFile fOpen1 («d:\\MasKurs\\konductor.txt», CFile:modeRead);

CString str;

CString str1;

while (fOpen. ReadString(str)!=0) {

m\_combo. AddString(str);

}

while (fOpen1. ReadString(str1)!=0) {

m\_combo1. AddString(str1);

}

 // fOpen. SeekToEnd();

 // fOpen. WriteString (m\_date+»\n»);

 // fOpen. WriteString (m\_number+»\n»);

 // fOpen. WriteString (m\_marsh+»\n»);

 // fOpen. WriteString («\*\*\*\*\*"+»\n»);

}

CString str1;

CString str2;

void path: OnSaveToPath()

{

UpdateData(TRUE);

CStdioFile fOpen («d:\\MasKurs\\path.txt», CFile:modeWrite);

CString str;

int index=m\_combo. GetCurSel();

int index1=m\_combo1. GetCurSel();

if (m\_date==»» || m\_number==»») {

MessageBox («Please fill all fields», «Info», MB\_OK);

return;

}

m\_combo. GetLBText (index, str1);

m\_combo1. GetLBText (index1, str2);

fOpen. SeekToEnd();

fOpen. WriteString («FIO driver «+str1+»\n»);

fOpen. WriteString («FIO konductor «+str2+»\n»);

fOpen. WriteString («Date «+m\_date+»\n»);

fOpen. WriteString («Number «+m\_number+»\n»);

fOpen. WriteString («\*\*\*\*\*\n»);

MessageBox («Saved to path.txt», MB\_OK);

}

void path: OnBeginMarshrute()

{

UpdateData(TRUE);

CStdioFile fOpen («d:\\MasKurs\\marshrutebegin.txt», CFile:modeWrite);

CString str;

if (m\_date1==»» || m\_number1==»» || m\_time==»») {

MessageBox («Please fill all fields», «Info», MB\_OK);

return;

}

fOpen. SeekToEnd();

fOpen. WriteString («Date «+m\_date1+»\n»);

fOpen. WriteString («Number «+m\_number1+»\n»);

fOpen. WriteString («Time «+m\_time+»\n»);

fOpen. WriteString («\*\*\*\*\*\n»);

MessageBox («Saved to marshrutebegin.txt», MB\_OK);

}

void path: OnSaveToMarshruteOkonchanie()

{

UpdateData(TRUE);

CStdioFile fOpen («d:\\MasKurs\\marshruteokonchanie.txt», CFile:modeWrite);

CString str;

if (m\_date2=»» || m\_number2==»» || m\_time1==»») {

MessageBox («Please fill all fields», «Info», MB\_OK);

return;

}

fOpen. SeekToEnd();

fOpen. WriteString («Date «+m\_date2+»\n»);

fOpen. WriteString («Number «+m\_number2+»\n»);

fOpen. WriteString («Time «+m\_time1+»\n»);

fOpen. WriteString («\*\*\*\*\*\n»);

MessageBox («Saved to marshruteokonchanie.txt», MB\_OK);

}

**Uch class**

void uch: OnSaveToUchet()

{

UpdateData(TRUE);

CStdioFile fOpen («d:\\MasKurs\\uchet.txt», CFile:modeWrite);

CString str;

fOpen. SeekToEnd();

fOpen. WriteString («Date «+m\_date+»\n»);

fOpen. WriteString («Number of tickets «+m\_number+»\n»);

fOpen. WriteString («Nomer «+m\_marsh+»\n»);

fOpen. WriteString («\*\*\*\*\*\n»);

MessageBox («Saved to uchet.txt», MB\_OK);

}