МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ (ФИЛИАЛ)
ФЕДЕРАЛЬНОГО ГОСУДАРСТВЕННОГО БЮДЖЕТНОГО
ОБРАЗОВАТЕЛЬНОГО УЧРЕЖДЕНИЯ ВЫСШЕГО ОБРАЗОВАНИЯ
«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
В Г. ТАГАНРОГЕ РОСТОВСКОЙ ОБЛАСТИ
ПИ (филиал) ДГТУ в г. Таганроге

ЦМК «Прикладная информатика»

Практикум

По выполнению практических работ № 1 - 39
по дисциплине
МДК.05.02 Разработка кода информационных систем
для специальности 09.02.07 Информационные системы и программирование,
квалификации
«Специалист по информационным системам»

Таганрог

2023

Составители: Е.В. Михайлович

Практикум по выполнению практических работ по дисциплине «МДК.05.02 Разработка кода информационных систем». ПИ (филиала) ДГТУ в г.Таганроге, 2023г.

В практикуме кратко изложены теоретические вопросы, необходимые для успешного выполнения практических работ, рабочее задание и контрольные вопросы для самопроверки.

Предназначено для обучающихся по специальности 09.02.07 «Информационные системы и программирование». Квалификации выпускника: «Специалист по информационным системам»

Ответственный за выпуск:

Председатель ЦМК: _____ О.В. Андриян

СОДЕРЖАНИЕ

1. Введение

В учебно-методическом пособии к практикуму по курсу «Проектирование и разработка информационных систем» изложены сведения, необходимые для успешного выполнения практических занятий по данному курсу. Описан процесс работы с инструментарием, применяемым на практических занятиях, представлен ряд типичных задач и подходы к их решению. Практические занятия посвящены углубленному знакомству обучающихся с управлением процесса разработки приложений с использованием инструментальных средств; обеспечением сбора данных для анализа использования и функционирования информационной системы; программированием в соответствии с требованиями технического задания; использованием критериев оценки качества и надежности функционирования информационной системы; применением методики тестирования разрабатываемых приложений; определением состава оборудования и программных средств разработки информационной системы; разработкой документации по эксплуатации информационной системы; проведением оценки качества и экономической эффективности информационной системы в рамках своей компетенции; модификацией отдельных модулей информационной системы..

Цель настоящего пособия – помочь обучающимся при выполнении практических работ, выполняемых для закрепления знаний по теоретическим основам и получения практических навыков работы на компьютерах.

Обучающийся должен знать: основные виды и процедуры обработки информации, модели и методы решения задач обработки информации; основные платформы для создания, исполнения и управления информационной системой; основные процессы управления проектом разработки; основные модели построения информационных систем, их структуру, особенности и области применения; методы и средства проектирования, разработки и тестирования информационных систем; систему стандартизации, сертификации и систему обеспечения качества продукции.

Обучающийся должен уметь: осуществлять постановку задач по обработке информации; проводить анализ предметной области; осуществлять выбор модели и средства построения информационной системы и программных средств; использовать алгоритмы обработки информации для различных приложений; решать прикладные вопросы программирования и языка сценариев для создания программ; разрабатывать графический интерфейс приложения; создавать и управлять проектом по разработке приложения; проектировать и разрабатывать систему по заданным требованиям и спецификациям.

Данное учебно-методическое пособие предназначено для обучающихся 3 и 4 курсов.

2. Правила выполнения практических занятий

Практические занятия выполняются каждым обучающимся самостоятельно в полном объеме и согласно содержанию методических указаний.

Перед выполнением обучающийся должен отчитаться перед преподавателем за выполнение предыдущего занятия (сдать отчет).

Обучающийся должен на уровне понимания и воспроизведения предварительно усвоить необходимую для выполнения практических занятий теоретическую и информацию.

Обучающийся, получивший положительную оценку и сдавший отчет по предыдущему практическому занятию, допускается к выполнению следующему занятию.

Обучающийся, пропустивший практическое занятие по уважительной либо неуважительной причине, закрывает задолженность в процессе выполнения последующих практических занятий.

3. Практические работы по дисциплине МДК.05.02 Разработка кода информационных систем

Материально техническое обеспечение практических работ

Реализация рабочей программы предполагает наличие компьютерного класса.
Оборудование учебного кабинета и рабочих мест:
1) рабочее место для преподавателя;
2) столы, стулья на 25-30 обучающихся;
Технические средства обучения:
1) Проектор / BENQ MX506, экран для проектора / CACTUS Wallscreen CS-PSW-206x274,274x206 см,4:3, настенно-потолочный, белый
2) Персональные компьютеры с программным обеспечением:

- 7-Zip 1602
- Adobe PDF Reader 11.0
- Google Chrome
- Notepad++ 6.9.2
- OpenOffice
- Openproj 1.4
- VirtualBox 5.1.12
- Microsoft Office Pro 2016
- Windows 10

ПРАКТИЧЕСКАЯ РАБОТА № 1
ПОСТРОЕНИЕ ДИАГРАММЫ ВАРИАНТОВ
ИСПОЛЬЗОВАНИЯ И ДИАГРАММЫ ПОСЛЕДОВАТЕЛЬНОСТИ И ГЕНЕРАЦИЯ КОДА

Цель работы: ознакомиться с методологией моделирования информационных систем на основе языка UML.

Рабочее задание

−выполнить задание;
−показать преподавателю;
−ответить на вопросы преподавателя.
Время выполнения: 2 ч

Теоретическая часть

Универсальный язык моделирования UML. Понятие диаграммы.
Виды диаграмм.
Основные элементы диаграммы вариантов использования. Основные элементы диаграммы последовательности.

Порядок выполнения работы

Задание № 1. Ознакомиться с методологией построения диаграммы вариантов использования основе языка UML.
Задание № 2. Проанализируйте пример построения диаграммы вариантов использования.
Пример. Магазин видеопродукции
Магазин продает видеокассеты, DVD-диски, аудиокассеты, CD-диски и т.д., а также предлагает широкой публике прокат видеокассет и DVD-дисков.
Товары поставляются несколькими поставщиками. Каждая партия товара предварительно заказывается магазином у некоторого поставщика и доставляется после оплаты счета. Вновь поступивший товар маркируется, заносится в базу данных и затем распределяется в торговый зал или прокат.
Видеоносители выдаются в прокат на срок от 1 до 7 дней. При прокате с клиента взимается залоговая стоимость видеоносителя. При возврате видеоносителя возвращается залоговая стоимость минус сумма за прокат. Если возврат задержан менее чем на 2 дня, взимается штраф в размере суммы за прокат за 1 день* кол-во дней задержки. При задержке возврата более чем на 2 дня – залоговая сумма не возвращается. Клиент может взять одновременно до 4 видеоносителей (прокат-заказ). На каждый видеоноситель оформляетсяквитанция.
Клиенты могут стать членами видео-клуба и получить пластиковые карточки. С членов клуба не берется залог (за исключением случая описанного ниже), устанавливается скидка на ставку проката и покупку товаров. Члены клуба могут делать предварительные заказы на подбор видеоматериалов для проката или покупки.
Каждый член клуба имеет некоторый стутус. Первоначально – "новичок". При возврате всрок 5 прокат-заказов, статус меняется на "надежный". При задержке хотя бы одного видеоносителя более чем на 2 дня, статус "новичок" или "надежный" меняется на "ненадежный" и клиенту высылается предупреждение. При повторном нарушении правил статус меняется на "нарушитель". Члены клуба со статусом "надежный" могут брать до 8 видеоносителей единовременно, все остальные – 4. С членов клуба со статусом "нарушитель" берется залоговая сумма.
Клиенты при покупке товара или получении видеоносителя в прокат могут расплачиваться наличными или кредитной картой.
Прокатные видеоносители через определенное количество дней проката списываются и утилизируются по акту. Списываются также товары и прокатные видеоносители, у которых обнаружился брак.
На рисунке 1 приведена диаграмма прецедентов для рассматриваемого примера. В этом примере можно выделить следующие субъекты и соответствующие им прецеденты:

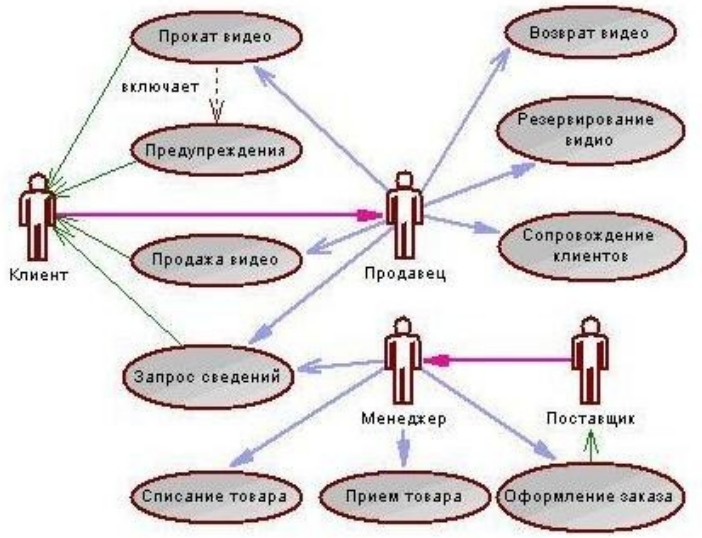


Рисунок 1

Менеджер изучает рынок видеопродукции, анализирует продажи (прецедент "Запрос сведений"), работает с поставщиками: составляет заявки на поставки товара (прецедент "Оформление заказа"), оплачивает и принимает товар (прецедент "Прием товара"), списывает товар (прецедент "Списаниетовара").

Продавец – работает с клиентами: продает товар (прецедент "Продажа видео"), оформляет членство в клубе (прецедент "Сопровождение клиентов"), резервирует (прецедент "Резервирование видио"), выдает в прокат (прецедент "Прокат видео") и принимает назад видеоносители (прецедент "Возврат видео"), отвечает на вопросы клиента (прецедент "Запрос сведений").

Поставщик – оформляет документы для оплаты товара (прецедент "Оформление заказа"), поставляет товар (прецедент "Прием товара"))

Клиент – покупает(прецедент "Продажа видео"), берет на прокат и возвращает видеоносители (прецеденты "Прокат видео" и "Возврат видео"), вступает в клуб (прецедент "Сопровождение клиентов"), задает вопросы (прецедент "Запрос сведений").

Последние два субъекта Поставщик и Клиент не будут иметь непосредственного доступа к разрабатываемой системе (второстепенные субъекты), однако именно они являются основным источником событий, инициализирующих прецеденты, и получателями результата работы прецедентов

От прецедента "Прокат видео" к прецеденту "Предупреждения" установлено отношение включения на том основании, что каждый выданный видеоноситель должен быть проверен на своевременный возврат и, в случае необходимости, выдано предупреждение клиенту.

Дальнейшее развитие модели поведения системы предполагает спецификацию прецедентов. Для этого традиционно используют два способа. Первый – описание с помощью текстового документа. Такой документ описывает, что должна делать система, когда субъект инициировал прецедент. Типичное описание содержит следующие разделы:

– краткоеописание;
– участвующиесубъекты;
– предусловия, необходимые для инициированияпрецедента;
– поток событий (основной и, возможно, подпотоки,альтернативный);
– постусловия, определяющие состояние системы, по достижении которого прецедент завершается.

Описательная спецификация прецедента "Прокат видео"

Раздел	Описание
Краткоеописание	Клиент желает взять на прокат видеокассету или диск, которые снимаются с полки магазина или были предварительно зарезервированы клиентом. При условии, что у клиента нет невозвращенных в срок видеоносителей, сразу после внесения платы фильм выдается напрокат. Если невозвращенные в срок видеоносители есть, клиенту выдается напоминание о просроченном возврате
Субъекты	Продавец, Клиент

Предусловия	В наличие имеются видеокассеты или диски, которые можно взять напрокат. У клиентов есть клубные карточки. Устройство сканирования работает правильно. Работники за прилавком знают, как обращаться с системой
Основнойпоток	Клиент может назвать номер заказа или взять видеоноситель с полки. Видеоноситель и членская карточка сканируются, и продавцу не сообщается никаких сведений о задержках, так, что он не задает клиенту соответствующих вопросов. Если клиент имеет статус <надежный>, он может взять до 8 видеоносителей, во всех остальных случаях – до 4-х. Если статус клиента определен как <нарушитель>, его просят внести задаток. Клиент расплачивается наличными или кредитной картой. После получения суммы, информация о наличии фильмов обновляется и видеоносители передаются клиенту вместе с квитанциями на прокат. О прокате каждого видеоносителя делается отдельная запись с указанием идентификационного номера клиента, даты проката, даты возврата, идентификационного номера продавца, полученнойсуммы. Прецедент генерирует предупреждения о просроченном возврате клиенту, если видеофильм не был возвращен в течение двух дней по истечении даты возврата и изменяет статус клиента на <ненадежный> (первое нарушение) или <нарушитель> (повторное нарушение)
Альтернативныйпоток	У клиента нет членской карточки. В этом случае прецедент <Сопровождение клиента> может быть активизирован для выдачи новой карточки. Видеофильмы не выдаются, поскольку у клиента есть невозвращенные в срок видеоносители. Попытка взять напрокат слишком много видеоносителей. Видеоноситель или кредитная карта не могут быть отсканированы из-за их повреждения У клиента не хватило наличных или платеж по кредитной карте отклонен
Постусловия	Видеофильмы сданы напрокат, и база данных соответствующим образом обновлена

Задание № 3. Постройте диаграмму вариантов использования для выбранной информационной системы (практическая работа №11).

Задание № 4. Ознакомиться с методологией построения диаграммы последовательности основе языка UML.

Задание № 5. Проанализируйте пример построения диаграммы последовательности (рисунок 2).

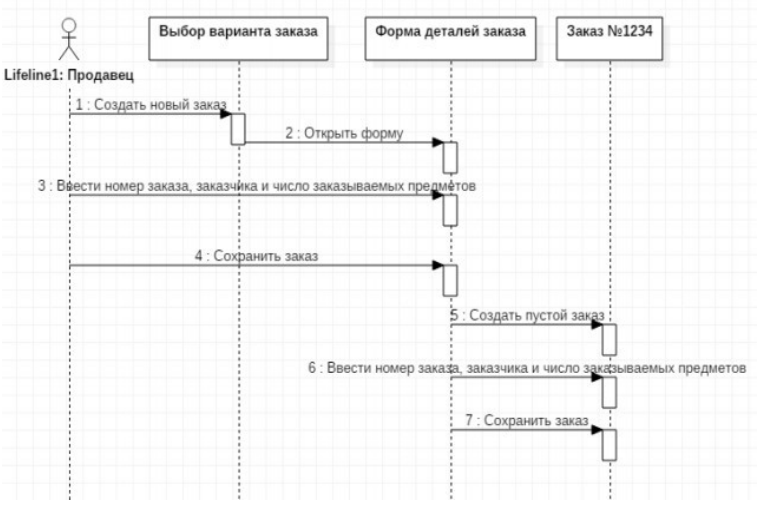


Рисунок 2

Пример

Вводзаказа.

Действующее лицо«Продавец». Сообщения:

– создать новыйзаказ;
– открытьформу;
– ввести номер заказа, заказчика и число заказываемыхпредметов;
– сохранитьзаказ;
– создать пустойзаказ;
– ввести номер заказа, заказчика и число заказываемыхпредметов;
– сохранитьзаказ.

Теперь нужно позаботиться об управляющих объектах и о взаимодействии с базой данных. Как видно из диаграммы, объект Форма Деталей Заказа имеет множество ответственностей, с которыми лучше всего мог бы справиться управляющий объект. Кроме того, новый заказ должен сохранять себя в базе данных сам. Вероятно, эту обязанность лучше было бы переложить на другойобъект.

Окончательный вид диаграммы последовательности представлен на рисунке 3.

Задание № 6. Постройте диаграмму последовательности для выбранной информационной системы (практическая работа № 11).

Задание № 7. Оформите отчет.

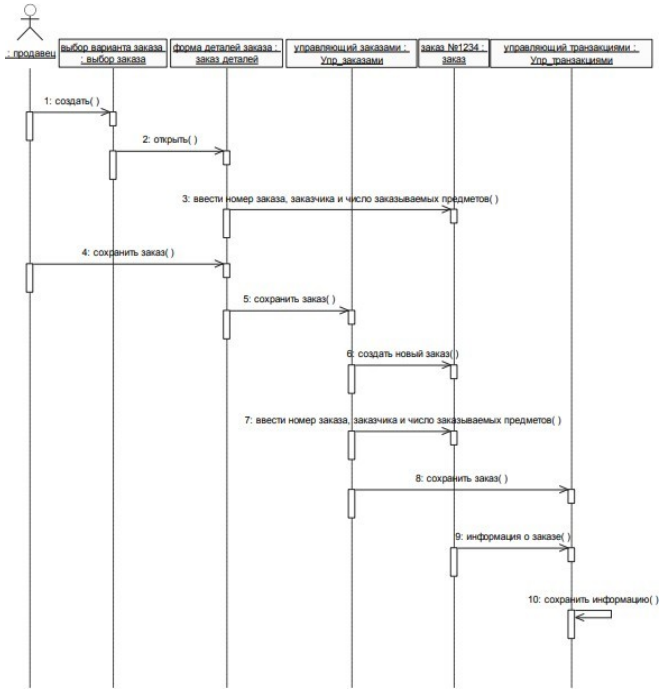


Рисунок 3

Контрольные вопросы

1. Какие основные диаграммы UML используются при моделировании информационных систем?
2. Каково назначение диаграммы классов в UML при моделировании информационных систем?
3. Какие сущности и связи описываются на диаграмме классов UML?
4. Что такое диаграмма вариантов использования UML и как она помогает при моделировании информационных систем?
5. В чем заключается роль диаграммы последовательности UML при моделировании информационных систем?
6. Какие диаграммы UML используются для описания поведения информационных систем?
7. Что такое диаграмма состояний UML и как она применяется при моделировании информационных систем?
8. Какие диаграммы UML помогают описать структуру базы данных в информационной системе?
9. Чем отличаются диаграмма компонентов и диаграмма развёртывания в UML и в чём состоит их роль при моделировании информационных систем?
10. Каким образом UML поддерживает моделирование информационных систем времени выполнения?

ПРАКТИЧЕСКАЯ РАБОТА № 2
НИЕ ДИАГРАММЫ КООПЕРАЦИИ И ДИАГРАММЫ РАЗВЕРТЫВАНИЯ И ГЕНЕРАЦИЯКОДА

Цель работы: ознакомиться с методологией моделирования информационных систем на основе языка UML.

Рабочее задание

−выполнить задание;
−показать преподавателю;
−ответить на вопросы преподавателя.
Время выполнения: 2 ч

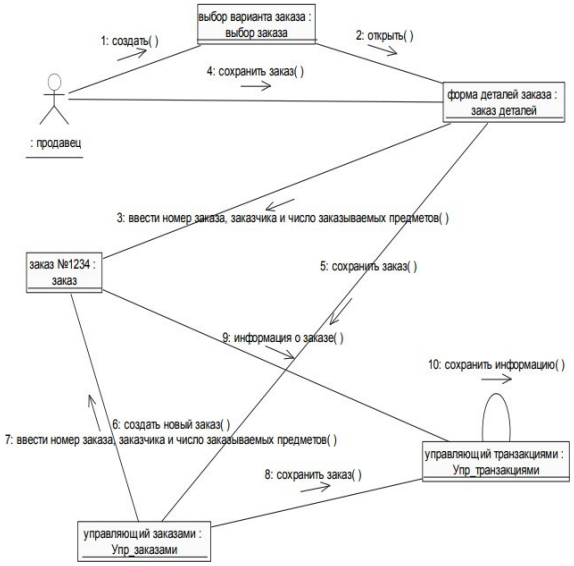
Теоретическая часть

Универсальный язык моделирования UML. Понятие диаграммы.
Виды диаграмм.
Основные элементы диаграммы кооперации. Основные элементы диаграммы развертывания.

Порядок выполнения работы

Задание № 1. Ознакомиться с методологией построения диаграммы кооперации основе языка UML.
Задание № 2. Проанализируйте пример построения диаграммы кооперации (рисунок 4).

Рисунок 4



Задание № 3. Постройте диаграмму кооперации для выбранной информационной системы (практическая работа № 11).
Задание № 4. Ознакомиться с методологией построения диаграммы развертывания основе языка UML.
Задание № 5. Проанализируйте пример построения диаграммы развертывания.
Примеры построения диаграмм развертывания
Фрагмент диаграммы развертывания с соединениями между узлами показан на рисунке 5.

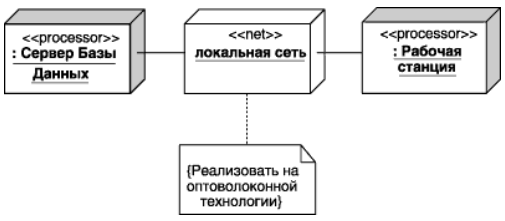


Рисунок 5

Диаграмма развертывания с отношением зависимости между узлом и развернутыми на нем компонентами приведена на рисунке 6.

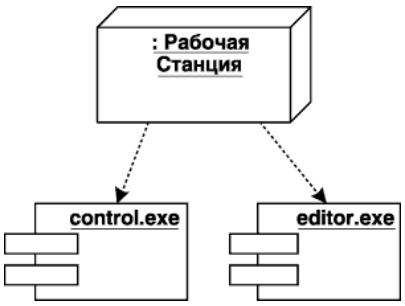


Рисунок 6

Диаграмма развертывания для системы мобильного доступа к корпоративной базе данных изображена на рисунке 7.

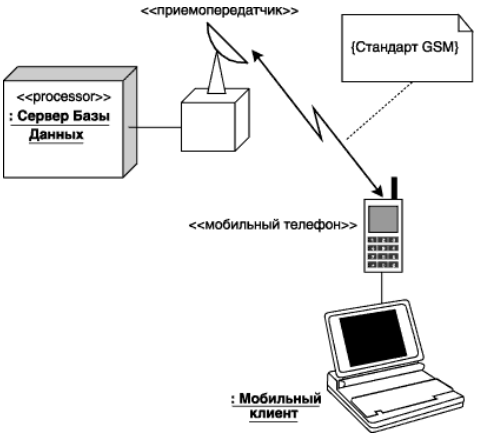


Рисунок 7

Задание № 6. Постройте диаграмму развертывания для выбранной информационной системы (практическая работа №11).
Задание № 7. Оформите отчет.

Контрольные вопросы

1. Какое место занимает UML в методологиях моделирования информационных систем?
2. Какие типы диаграмм UML наиболее часто применяются при моделировании информационных систем?
3. Каким образом диаграмма классов UML помогает описать структуру информационной системы?
4. Какие основные элементы и связи используются на диаграмме последовательности UML для моделирования работающей системы?
5. Как диаграмма случаев использования UML помогает описать функциональные требования информационной системы?
6. В чем заключается роль диаграммы активностей UML при моделировании бизнес-процессов в информационной системе?
7. Как диаграмма компонентов UML помогает описать архитектуру информационной системы?
8. Что такое диаграмма развертывания UML и как она применяется при моделировании информационных систем?
9. Каким образом диаграмма состояний UML помогает моделировать поведение объектов в информационной системе?
10. Какие инструменты и среды разработки поддерживают использование UML при моделировании информационных систем?

ПРАКТИЧЕСКАЯ РАБОТА №3
Построение диаграммы Деятельности, диаграммы Состояний и диаграммы Классов и генерация кода

Цель работы: научиться строить диаграмму вариантов использования.

Рабочее задание
−выполнить задание;
−показать преподавателю;
−ответить на вопросы преподавателя.
Время выполнения: 2 ч

Теоретическая часть:
Визуальное моделирование в UML можно представить как некоторый процесс поуровневого спуска от наиболее обшей и абстрактной концептуальной модели исходной системы к логической, а затем и к физической модели соответствующей программной системы. Для достижения этих целей вначале строится модель в форме так называемой диаграммы вариантов использования (usecasediagram), которая описывает функциональное назначение системы или, другими словами, то, что система будет делать в процессе своего функционирования. Диаграмма вариантов использования является исходным концептуальным представлением или концептуальной моделью системы в процессе ее проектирования и разработки.
Разработка диаграммы вариантов использования преследует цели:
- Определить общие границы и контекст моделируемой предметной области на начальных этапах проектирования системы.
- Сформулировать общие требования к функциональному поведению проектируемой системы.
- Разработать исходную концептуальную модель системы для ее последующей детализации в форме логических и физических моделей.
- Подготовить исходную документацию для взаимодействия разработчиков системы с ее заказчиками и пользователями.

Суть данной диаграммы состоит в следующем: проектируемая система представляется в виде множества сущностей или актеров, взаимодействующих с системой с помощью так называемых вариантов использования. При этом актером (actor) или действующим лицом называется любая сущность, взаимодействующая с системой извне. Это может быть человек, техническое устройство, программа или любая другая система, которая может служить источником воздействия на моделируемую систему так, как определит сам разработчик. В свою очередь, вариант использования (usecase) служит для описания сервисов, которые система предоставляет актеру. Другими словами, каждый вариант использования определяет некоторый набор действий, совершаемый системой при диалоге с актером. При этом ничего не говорится о том, каким образом будет реализовано взаимодействие актеров с системой.

Состав диаграммы UseCase

Диаграмма вариантов использования состоит из актеров, для которых система производит действие и собственно действия UseCase, которое описывает то, что актер хочет получить от системы. Актер обозначается значком человечка, а UseCase - овалом. Дополнительно в диаграммы могут быть добавлены комментарии.

Виды взаимодействий

Между актерами и вариантами использования могут быть различные виды взаимодействия. Основные виды взаимодействия следующие:
- **Простая ассоциация** - отражается линией между актером и вариантом использования (без стрелки). Отражает связь актера и варианта использования. На рисунке между актером администратор и вариантом использованияпросматривать заказ.

- **Направленная ассоциация** - то же что и простая ассоциация, но показывает, что вариант использования инициализируется актером. Обозначается стрелкой.
- **Наследование** - показывает, что потомок наследует атрибуты и поведение своего прямого предка. Может применяться как для актеров, так для вариантов использования.
- **Расширение** (extend) - показывает, что вариант использования расширяет базовую последовательность действий и вставляет собственную последовательность. При этом в отличие от типа отношений "включение" расширенная последовательность может осуществляться в зависимости от определенных условий.
- **Включение** (include) - показывает, что вариант использования включается в базовую последовательность и выполняется всегда (на рисунке не показан).

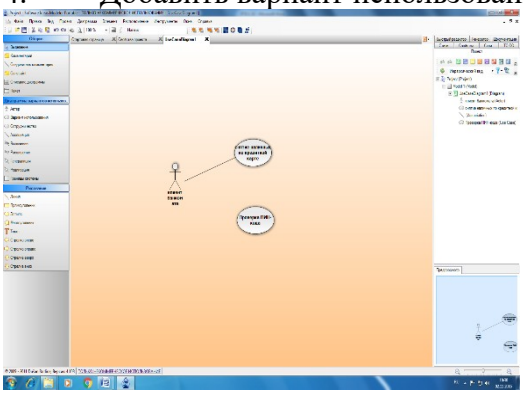
Существуют и другие виды взаимодействия, но они менее важны и реже применяются.

Порядок выполнения работы

Задание 1. Построить диаграмму вариантов использования модели вариантов использования банкомата.

Выполните следующие действия:

1. Добавить актера с именем Клиент банкомата.
2. Добавить вариант использования Снятие наличных по кредитной карте
3. Добавить направленную ассоциацию от бизнес-актера Клиент Банкомата к варианту использования Снятие наличных по кредитной карте
4. Добавить вариант использования Проверка ПИН-кода.



5. Добавить актера с именем Банк.
6. Добавить вариант использования Получение справки о состоянии счета.
7. Добавить вариант использования Блокирование кредитной карточки.
8. Добавить направленную ассоциацию от бизнес-актера Клиент Банкомата к варианту использования Получение справки о состоянии счета.
9. Добавить направленную ассоциацию от варианта использования Снятие наличных по кредитной карточке к сервису Банк.
10. Добавить направленную ассоциацию от варианта использования Получение справки о состоянии счета к сервису Банк.
11. Добавить отношение зависимости со стереотипом «include», направленное от варианта использования Снятие наличных по кредитной карте к варианту использования Проверка Пин-кода.
12. Добавить отношение зависимости со стереотипом «include», направленное от варианта использования Получение справки о состоянии счета к варианту использования Проверка Пин-кода.
13. Добавить отношение зависимости со стереотипом «extend», направленное от варианта использования Блокирование кредитной карточки к варианту использования Проверка Пин-кода.

Задание 2. Построить диаграмму вариантов использования.

Имеются следующие данные:

- четыре действующих лица: Клиента банка, Банк, Кассира и Оператора,
- пять вариантов использования: Снять наличные, Перевести деньги со счета, Положить деньги на счет, Пополнить запас денег и Подтвердить пользователя,
- три зависимости, и отношения между действующими лицами и вариантами использования.

Варианты использования: Снять наличные, Перевести деньги со счета, Положить деньги на счет - требуют включения идентификации клиента в системе. Это поведение может быть выделено в новый вариант использования включения, называемый Подтвердить пользователя. Базовые варианты использования не зависимы от метода, используемого для идентификации. Поэтому он инкапсулируется (скрывается) в варианте использования включения. С точки зрения базовых вариантов использования не имеет значение производится ли идентификация с помощью магнитной карты или сканированием сетчатки глаза. Они только зависят от результата выполнения варианта использования Подтвердить клиента.

Задание для самостоятельной работы: Построить диаграмму вариантов использования на основе вербальной модели информационной системы «Компьютерный клуб»

Контрольные вопросы:

1. Какие цели преследует разработка диаграммы использования?
2. Для чего нужна диаграмма вариантов использования?
3. Из чего состоит диаграмма вариантов использования?
4. Виды взаимодействия используемые в диаграмме вариантов использования?
5. Из чего состоит созданная вами диаграмма?

ПРАКТИЧЕСКАЯ РАБОТА № 4
СТРОЕНИЕ ДИАГРАММЫ КОМПОНЕНТОВ И ГЕНЕРАЦИЯ КОДА

Цель работы: ознакомиться с методологией моделирования информационных систем на основе языка UML.

Рабочее задание:

−выполнить задание;
−показать преподавателю;
−ответить на вопросы преподавателя.

Время выполнения: 2 ч

Теоретическая часть

Универсальный язык моделирования UML. Понятие диаграммы.
Виды диаграмм.
Основные элементы диаграммы компонентов. Основные элементы диаграммы развертывания.

Порядок выполнения работы

Задание № 1. Ознакомиться с методологией построения диаграммы компонентов основе языкаUML.
Задание № 2. Проанализируйте пример построения диаграммы компонентов. Выделяем компоненты, отображаем зависимости между ними.
Фрагмент диаграммы компонентов с отношениями зависимости и реализации показан на рисунке8.
Графическое изображение отношения зависимости между компонентами приведено на рисунке9.
На рисунке 10 показано графическое изображение зависимости между компонентом и классами.

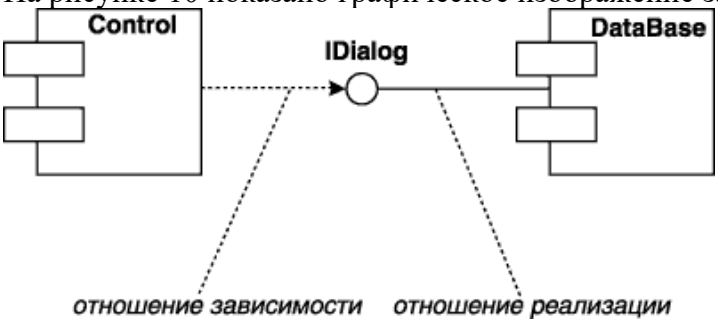


Рисунок8
Рисунок9

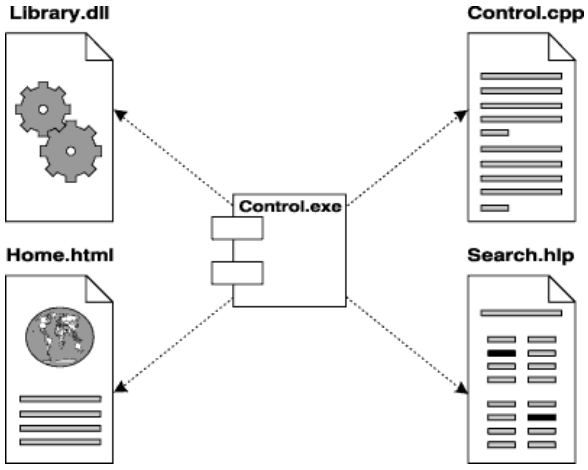


Рисунок 10

Задание № 3. Постройте диаграмму компонентов для выбранной информационной системы (практическая работа № 11).
Задание № 4. Оформите отчет.

Контрольные вопросы

1. Каким образом диаграмма компонентов UML помогает описать архитектуру информационной системы?
2. Какие элементы и связи используются на диаграмме компонентов UML?

3. Какие типы компонентов могут быть представлены на диаграмме компонентов UML?
4. Как диаграмма компонентов UML помогает визуализировать зависимости и взаимодействия между компонентами системы?
5. Какие дополнительные артефакты и свойства могут быть представлены на диаграмме компонентов UML?
6. В чем состоит роль артефактов на диаграмме компонентов UML и как они связаны с компонентами?
7. Каким образом можно представить зависимости и связи между компонентами на диаграмме компонентов UML?
8. Какие инструменты и среды разработки поддерживают создание и визуализацию диаграмм компонентов UML?
9. Какие преимущества и недостатки существуют при использовании диаграмм компонентов UML для описания архитектуры информационной системы?
10. Как включить информацию о интерфейсах и использовании компонентов в диаграмму компонентов UML?

Практическая работа №5
ОБОСНОВАНИЕ ВЫБОРА ТЕХНИЧЕСКИХ СРЕДСТВ

Цель работы:научиться определять необходимые технические средства.

Рабочее задание:
−выполнить задание;
−показать преподавателю;
−ответить на вопросы преподавателя.
Время выполнения: 2 ч

Теоретическая часть
На выбор комплекса технических средств разработанной системы влияет её функциональность и методы хранения информации.
На рисунке 1 приведен пример диаграммы модулей (фрагмент) для системы составления линейного кроссворда. В таблице 1 приведено описание модулей.

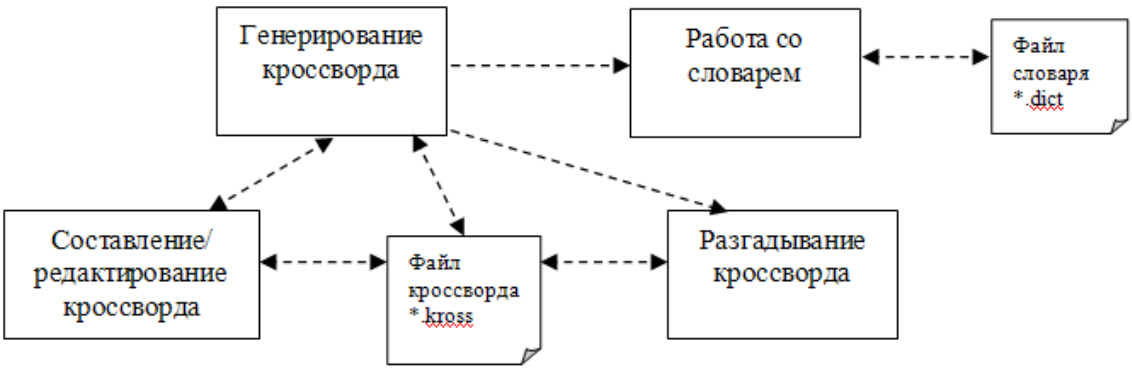


Рисунок 1 – Диаграмма модулей системы (фрагмент)

Таблица 1 – Описание модулей системы (фрагмент)

Название модуля	Название файла	Назначение	Подсистема
Генерирование кроссворда	Generation.c	Отвечает за автоматическое	Подсистема генерирования

		составление кроссворда при заданных параметрах	кроссворда
Работа со словарем	Dict.c	Отвечает за редактирование словаря с понятиями и их определениями	Подсистема работы со словарем
Составление/ редактирование кроссворда	Rewrite_Cross.c	Отвечает за редактирование кроссворда	Подсистема ручного составления
Разгадывание кроссворда	Unravel.c	Отвечает за разгадывание кроссворда	Подсистемаразгадывания кроссворда

Периферийная техника необходимая для решения задач, поставленных перед системой, должна выполнять следующие функции:

– ввод данных с клавиатуры;
– управление курсором манипулятором типа «мышь»;
– вывод информации на дисплей и на печать;
– передача информации в БД.

Порядок выполнения работы

Согласно вышеперечисленным требованиям и результатам расчетов быстродействия и ресурсов для нормального функционирования системы составить перечень необходимых технических средств.

Контрольные вопросы

1. Какая роль играет методология обоснования выбора технических средств при создании информационной системы?
2. Какие основные этапы включает процесс обоснования выбора технических средств для информационной системы?
3. Какие критерии следует учитывать при выборе технических средств для информационной системы?
4. Как проводится сравнительный анализ различных технических средств на основе заданных критериев?
5. Каким образом оцениваются затраты на внедрение и поддержку выбранных технических средств?
6. Какие риски могут возникнуть при выборе и использовании определенных технических средств?
7. Как влияют требования к безопасности на выбор технических средств для информационной системы?
8. Какие факторы следует учитывать при оценке совместимости выбранных технических средств с существующей инфраструктурой?
9. Каким образом проводится оценка достижимости заданных целей и требований с использованием выбранных технических средств?
10. Каким образом документируется и обосновывается выбор технических средств для информационной системы?

Практическая работа №6

Основы работы в среде Microsoft Visual Studio

Цель работы: знакомство с языком программирования C# и системой MS Visual Studio.

Теоретическая часть

Консоль – совокупность стандартных устройств ввода-вывода (клавиатура, монитор). Для работы с консолью предназначен класс **Console** в пространстве имен **System**. Основные методы: *Console.Write*, *Console.WriteLine*, *Console.Read*, *Console.ReadLine*.

Пусть, в программе заданы: *int d* = 48; *double y* = 5.7412; *string s* = "бит";

В простейшем выводе на консоль можно использовать конкатенацию, например:

```
Console.WriteLine("d = " + d + " " + s  + "    y = "  + y);
```

Будет выведено: `d = 48 бит    y = 5,7412`

Заметим, что в коде программы резделителем целой и дробной части деся- тичной дроби является **точка**, а при вводе и выводе разделитель определяется ло- кализацией операционной системы, т. е. для русифицированных ОС – **запятая**!

Форматный вывод – использование в строке вывода местозаполнителей (*placeholder*), которые включают параметры формата и управляющие символы:

{ номер [, к-во позиций] [: формат] }

Номера элементов в списке вывода могут идти не по порядку и повторять- ся. Количество позиций под выводимое значение может иметь знак «минус», тогда оно выравнивается внутри отведенного места по левому краю, иначе – по правому. Формат вывода обозначается латинскими буквами, например: ***F*** или ***f*** – количество десятичных цифр дробной части числа (***f2*** – две). В строке вывода могут использоваться управляющие символы (они предваряются косой чертой - слешем), например: ***n*** – переход на новую строку, ***t*** – табуляция. Заметим, что выводимую строку в кавычках разрывать нельзя, а список переменных после закрывающих кавычек можно записывать в новой строке:

```
Console.WriteLine("d = {0,6} {1,-8} y = {2} \n d = {0,-6} {1,-8}
y = {2:f2} ",  d, s, y);
```

Будет выведено:
```
d =     48 бит     y = 5,7412
d = 48     бит     y = 5,74
```

Начиная с версии 6.0 C# для вывода данных можно использовать *интерпо- ляцию строк*, размещая имена переменных прямо в строке, предварив ее сим- волом $, например:

```
Console.WriteLine($"d = {d} y = {s} \nd = {d} y = {y}");
```

Для совместимости с более старыми версиями во всех примерах данного пособия используется форматный вывод.

Для ввода данных с консоли используют методы:

ReadLine – возвращает строку типа *string* и *Read* – возвращает код символа. Для дальнейшей работы требуется преобразование в нужный тип!

Для этого используют: метод **Parse**, который выполняет разборку (пар- синг) строки или класс **Convert**, который содержит методы преобразования в требуемый тип.

```
char c = (char)Console.Read();      // ввод кода и преобразование в символ
string st = Console.ReadLine();     // ввод строки
int k = int.Parse(st);  или  int m = Convert.ToInt32(st);  // преобразование в целое
Console.WriteLine("строка st={0} число k={1}  m={2}", st, k, m);      // вывод
double x = double.Parse(st);        // преобразование в вещественный тип double
y = Convert.ToDouble(st);           // преобразование в вещественный тип
Console.WriteLine("строка st={0} число x={1}  y={2}", st, x, y); // вывод
```

Запись некоторых арифметических операций в языке C# отличается от принятой в математике: ***** умножение, **/** деление, **%** остаток от целочисленного деления.

Пример 1

Простейшее приложение: вывод на консоль приветствия.

1) Запустим **MS Visual Studio**.

2) На появившейся стартовой странице выберем **Создать проект** (New Project).

3) На следующей странице выберем тип приложения **Консольное** (Console Application) и шаблон **Visual C#**.

4) В поле ввода **Расположение** (Location) выберем или зададим рабочую папку, в которой будет сохраняться проект, например, **\rabota**.

5) Введем имя проекта и **Решения** (Solution), например, **con01.**

6) Откроется окно программного кода с шаблоном. В нем с помощью клю- чевых слов **using** объявлены пространства имен, из которых можно использо- вать стандартные классы непосредственно, без указания имени пространства (в нашем примере используется пространство имен **System**). С помощью клю- чевого слова **namespace** создано собственное пространство имен, имя которого совпадает с именем проекта **con01**.

7) Для упрощения шаблона можно сразу удалить «лишние» строки и вы- ражения, которые не будут использоваться в нашей программе, например, не-

используемые параметры метода **Main()** и пространства имен.

8) В теле метода **Main** шаблона наберем код программы:

```
using System; class
  Program
  {
   static void Main()
   {
     Console.Write("Введите имя:  ");                  // вывод приглашения string
     nam = Console.ReadLine();                         // ввод имени, тип string
     Console.WriteLine("Привет, " + nam + "!");        // вывод приветствия
          // в конец добавляем строку с методом ReadKey,
          // который заставляет программу ожидать нажатия любой клавиши
     Console.ReadKey();
   }
  }
```

9) После набора кода программа обязательно тестируется (меню **Отладка** (Debug) или клавиша *F5*). При этом будет создан исполняемый PE-файл (Portable Executable) с расширением **.exe** и помещен в папку **\bin\Debug** проек- та. Его можно переносить в другое место и запускать без системы Visual Studio при наличии виртуальной машины – общеязыковой среды исполнения CLR (**Common Language Runtime**) платформы .NET Framework.

10) Протестируем программу. В окне вывода получим:

```
Введите имя:  Маша
Привет, Маша!
```

11) Модифицируем программу, добавив условие: если не введено имя, то выводится строка «неизвестный»

```
if (nam == "")  nam = "неизвестный";                 // проверка условия
```

12) Протестируем окончательный вариант. Теперь без ввода имени получим:

```
Введите имя:
Привет, неизвестный!
```

Примечание. Свойства окна вывода (цвет фона, шрифт…) можно наст- раивать.

Пример 2

Приложение, которое по введенному радиусу вычисляет и выводит на консоль длину окружности и площадь круга

1) Создадим проект с именем **con02**.

2) В теле метода **Main** наберем код программы:

```
  static void Main()
   {
    Console.Write("Введите радиус:  ");
                        //  ввод и преобразование типа string в int
    int r = int.Parse(Console.ReadLine());
    double p = 2 * Math.PI*r;    double s = Math.PI*r*r;
                        // форматный вывод – три десятичных знака
    Console.WriteLine("Длина окружности {0:f3}, площадь круга {1:f3}", p, s);
    Console.ReadKey();
   }
```

3) Протестируем программу (*F5*). При необходимости откорректируем про- граммный код. Результат может выглядеть так:

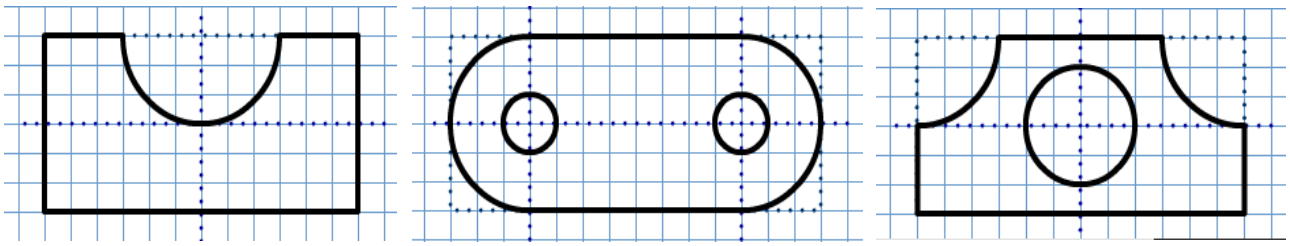
```
Введите радиус:  4
Длина окружности  25,133 , площадь круга  50,265
```

Порядок выполнения работы

Создайте консольные приложения, которые вычисляют и выводят:

Рабочее задание

1. Среднее арифметическое *sred* двух введенных чисел *a* и *b*.
2. Площадь *s* и периметр *p* прямоугольника по сторонам *a* и *b*.
3. Площадь поверхности $s = 4 \pi r^2$ и объем шара $v = 4/3 \pi r^3$ по радиусу *r*.
4. Значение функции $z = x^2 + x y - y^2$ (ввод *x* и *y*).
5. Стоимость товара в трех валютах по его стоимости в бел. рублях (курсы вводятся с клавиатуры).
6. Стоимость поездки на автомобиле (ввод: *s* – расстояние, *b* – расход бен- зина на 100 км, *c* - цена бензина за 1 литр).
7. Смещение от положения равновесия точки, совершающей гармониче- ские колебания $x = A \sin(w, t)$ (ввод: амплитуда A, угловая частота *w*, время *t*).
8. Расход материала *s* для изготовления детали по чертежу. Масштаб: 1 клетка = 4 см.



Контрольные вопросы

1. Как объявить и инициализировать переменную в C#?
2. Как объявить константу в C#?
3. Как объявить функцию (метод) в C#?
4. Как вызвать функцию (метод) в C#?
5. Как импортировать пространство имён (namespace) в C#?
6. Как создать новый проект в MS Visual Studio?
7. Как добавить новый файл в проект в MS Visual Studio?
8. Как использовать отладчик (debugger) в MS Visual Studio?
9. Как запустить программу в режиме отладки (debug mode) в MS Visual Studio?
10. Как скомпилировать и выполнить программу в MS Visual Studio?

Практическая работа №7
Алгоритмическая конструкция «ветвление»

Цель работы: формирование навыков реализации алгоритмов ветвления.

Теоретическая часть

Алгоритмическая конструкция **ветвление** используется для выбора одной из двух (развилка) или нескольких альтернатив (переключатель).

Развилка реализуется с помощью условного оператора **if: if** (условие) { блок_1
}
[**else** { блок_2 }]

Сначала проверяется условие. Если оно верно, выполняется блок_1, иначе выполняется блок_2.

В условиях используют операции сравнения, которые в языке C# записы- ваются так: = = равно, **!**= не равно, < , >, <=, >= ; а также логические операции && “и”, || “или”, **!** отрицание “не”.

Блоки могут содержать фрагменты кода, заключенные в фигурные скобки, которые можно опустить, если фрагмент состоит из одного выражения.

В случаях выбора и присваивания только одного значения из двух альтер- нативных вариантов вместо конструкции **if…else** удобно использовать тернар- ный оператор условного присваивания (**= ? :**):

= (условие) ? вариант1 : вариант2 ;

При необходимости выбора из нескольких вариантов используется кон- струкция **switch ...case** (оператор выбора или переключатель)**:**

```
switch (выражение)
{
  case выражение_1: блок_1 ;   break;
  case выражение_2: блок _2 ;   break;
  ……..
  case выражение_n: блок_n ;  break; [
  default: блоки_XXX; ]
}
```

При отладке программ часто используют случайные числа. Методы их ге- нерации описаны в классе **System.Random**. Для получения псевдослучайной последовательности чисел сначала создается экземпляр класса **Random**, затем методом **Next(n1,n2)** генерируется число в диапазоне [n1, n2], например:

```
Random rnd = new Random();  int n = rnd.Next(-20, 70);
```

Пример 1

*Использование конструкции **if … else** для проверки знака случайного числа и тернарного оператора для проверки четности.*

1. Создадим проект **con111**. В теле метода **Main** наберем код программы:

```
string mes1 = "", mes2 = "";          // объявление и инициализация
Random rnd = new Random();          // создаем экземпляр класса Random
int n = rnd.Next(-40, 40);         // генерируем случайное число от –40 до 40
if (n>=0) mes1 ="положительное";          // используем конструкцию if…else
    else mes1 = "отрицательное";
                                          // используем тернарный оператор
string mes2 = (n % 2 == 0) ? "четное" : "нечетное"; Console.WriteLine("Число
{0} {1} {2}", n, mes1, mes2); Console.ReadKey();
```

2. Протестируем программу. Откорректируем программный код.

Пример 2

*Использование конструкции выбора **switch…case**.*

1) Создадим проект **con112**. Наберем код программы в теле метода **Main**:

```
string mes;
Console.WriteLine("Введите день недели: Пн, Вт, Ср, Чт, Пт, Сб, Вс");
string day = Console.ReadLine();
  switch (day)
   { case "Сб": mes = "Иду в гости"; break; case
      "Вс": mes = "Отдыхаю"; break; default:
                 mes = "Работаю"; break;
   }
Console.WriteLine(mes);
Console.ReadKey();
```

2. Протестируем программу. Откорректируем программный код.

Пример 3

Проверка логина и пароля, введенных с клавиатуры.

Пусть эталонные логин длиной не меньше 6 символов и пароль хранятся в строковых переменных **myLog** и **myPas** в коде программы. Сначала проверя- ем длину и совпадение введенного логина с заданным. В случае правильности логина проверяем совпадение пароля

1. Создадим проект **con113**. Наберем код программы в теле метода **Main**:

```
    // задаем эталонные логин и пароль, а также выводимые сообщения
string  myLog = "qwerty",  myPas = "asdf";
string  mes = "", mesYes = "Добро пожаловать!",  mesNo = "Вход воспрещен!";
Console.Write("Введите логин:   ");
string log = Console.ReadLine();                          // вводим логин
    // проверяем длину и совпадение логина
if (log.Length < 6) mes = "Логин очень короткий!\n" + mesNo; else
   if (log != myLog)  mes = "Неверный логин!\n" + mesNo;
     else
    // в случае истинности логина вводим и проверяем пароль
     { Console.Write("Введите пароль:   ");
       string pas = Console.ReadLine();                   // вводим пароль

       mes = (pas == myPas) ? mesYes : "Неверный пароль!\n" + mesNo;
     }
Console.WriteLine(mes);                                   // выводим сообщения
Console.ReadKey();
```

2. Протестируем программу. Откорректируем программный код.

Пример 4

Простейший калькулятор на 4 действия.

1. Создадим проект **con114**. Наберем код программы в теле метода **Main**:

```
// инициализируем переменные res и ok. double A,
B, res = 0; bool ok = true;
Console.Write("Введите число A:  ");
A = double.Parse(Console.ReadLine());      // ввод строки и преобразование
Console.Write("Введите число B:  "); B =
double.Parse(Console.ReadLine());
Console.Write("Введите знак операции (+-*/) "); string
op = Console.ReadLine();
  switch (op)
  { case "+": res = A + B; break; case
     "-": res = A - B; break; case "*":
     res = A * B; break; case "/": res
     =  A  /  B;  break;  default:  ok  =
     false; break;}
if (ok)
    Console.WriteLine("{0} {1} {2} = {3}", A, op, B, res); // вывод else
Console.WriteLine("Недопустимая операция"); Console.ReadKey();
```

2. Протестируем программу. Откорректируем программный код.

В рассмотренном примере простейшего калькулятора 4 полезно произво- дить проверку вводимых данных и, если введено не число, присваивать значе- ние по умолчанию, например 1.

Фрагмент кода с проверкой и преобразованием строки в число примет вид:

```
if (!double.TryParse(Console.ReadLine(), out A))
    {  Console.WriteLine("Не число!");  A = 1;  }
if (!double.TryParse(Console.ReadLine(), out B))
    {  Console.WriteLine("Не число!");  B = 1;  }
```

3. Протестируем окончательный вариант.

Порядок выполнения работы

Создайте консольные приложения, в которых выполняются заданные дей- ствия:

Рабочее задание

1. Проверяется делимость введенного целого числа **n** на **d** (ввод: число **n**, делитель **d**; оператор **if**).

2. По введенному номеру месяца выводится название поры года (зима, вес- на, лето, осень) и сообщение: сессия, каникулы, 1 семестр, 2 семестр (**if**).

3. Проверяется соответствие веса и роста (ввод рост и вес; вывод одного из сообщений «Норма», «Нужно похудеть», «Нужно поправиться», оператор **if**). Нормальный вес (в кг) = (рост (в см) – 100) ± 10 %.

4. Выводится название предмета по введенной первой букве: ф – физика, м – математика, и – история, г – география, б – биология (оператор **switch**).

5. Выводится название страны и ее столицы по введенной первой букве: б – Беларусь, Минск, р – Россия, Москва, л – Литва, Вильнюс (**switch**).

6. Выводится название дня недели по введенному номеру (1 – пн, 2 – вт, …) сообщение «рабочий день» или «выходной» (операторы **switch, if**).

7. Введенная цифра (от 0 до 5) выводится прописью (оператор **switch**).

8. Проверяется правильность логина строго из 5 букв и пароля из 6 цифр.

Контрольные вопросы

1. Как реализовать условное выражение if-else в C#?
2. Как реализовать множественный выбор (switch) в C#?
3. Как сравнить два значения и проверить их равенство в C#?
4. Как сравнить два значения и проверить, что одно больше другого, в C#?
5. Как проверить, выполняются ли сразу несколько условий (логическое И) в C#?
6. Как проверить, выполняется ли хотя бы одно из нескольких условий (логическое ИЛИ) в C#?
7. Как реализовать условное выражение if-else-if в C#?
8. Как использовать тернарный оператор для реализации простых ветвлений в C#?
9. Как проверить, является ли значение переменной null в C#?
10. Как использовать блок try-catch для обработки исключений ветвления в C#?

Практическая работа №8
Алгоритмическая конструкция «цикл»

Цель работы: формирование навыков реализации циклических алгоритмов.

Теоретическая часть

При необходимости многократного использования фрагмента программно- го кода используют алгоритмическую конструкцию **цикл** (повторение).

Возможны несколько вариантов реализации циклов:

– с предусловием – с помощью оператора **while;**

– с постусловием – с помощью оператора **do … while;**

– с параметром – с помощью оператора **for;**

– перебора элементов массива – с помощью оператора **foreach … in.**

При реализации циклов часто используют следующие операции:

++ инкремент, – – декремент (увеличение и уменьшение на 1); присваива- ние с операциями +=, – =, *=, /=. Например, k++ означает k = k + 1, а вместо z = z – 5 пишут z – = 5 (без пробела между – и =).

Используются также операторы передачи управления:

– **break** прерывает выполнение цикла и инициирует выход из блока;

– **continue** выполняет переход к следующей итерации внутри цикла;

– **return** завершает выполнение функции и передает управление в точку ее вызова;

– **goto** выполняет безусловную передачу управления.

Пример 1

*Использование цикла for для вычисления суммы и произведения **n** натуральных чисел.*

1. Создадим проект **con121**. Наберем код программы в теле метода **Main**:

```
Console.Write("Введите целое число (от 3 до 9)   ");
            // ввод строки, преобразование в int
int n = int.Parse(Console.ReadLine());
            // задание начальных значений sum и pro перед телом цикла
int sum = 0;   int pro = 1; for
(int i=1; i<=n; i++)
 { sum += i;  pro *= i;
            // тело цикла:  вычисления и вывод
   Console.WriteLine("шаг={0} сумма = {1} произведение = {2}", i, sum, pro);
 }
Console.ReadKey();
```

2. Протестируем программу. Откорректируем программный код.

Пример 2

Использование цикла while. За один день уровень радиации уменьшается на 5 %. За сколько дней он уменьшится вдвое?

1. Создадим проект **con122**. Наберем код программы в теле метода **Main**:

```
Console.Write("Введите начальный уровень радиации(100 – 800 единиц:"); double
rad0 = double.Parse(Console.ReadLine());
    // задание начальных значений  rad и day перед телом цикла
double rad=rad0; int day=0;
while (rad>rad0/2)            // проверка условия уменьшения радиации вдвое
 { rad *= 0.95; day++;        // ежедневное уменьшение радиации на 5 %
   Console.WriteLine("День {0}, радиация = {1}", day, rad);
 }
Console.ReadKey();
```

2. Протестируем программу. Откорректируем программный код.

Пример 3

*Использование циклов **while** и **do…while** для моделирования движения те- ла. Тело брошено вертикально вверх с начальной скоростью **Vo**. Найти макси- мальную высоту подъема и время полета.*

Будем считать тело материальной точкой, не учитывать сопротивление воздуха и пренебрегать изменением ускорения свободного падения с высотой. Выберем начало вертикальной оси координат 0Y в точке бросания тела. Свя- жем начало отсчета времени с моментом бросания.

Уравнения движения в проекции на выбранную ось имеют вид:

$$y = Vo\ t - g\ t^2/2; \quad V = Vo - g\ t.$$

Алгоритм решения задачи заключается в использовании исходных уравне- ний движения в циклах с достаточно малым шагом изменения времени, напри- мер, t += 0.001. Максимальная высота подъема определяется в цикле while при условии, что скорость V >= 0, а время полета в цикле do … while, который выполняется до тех пор, пока координата y > 0.

1. Создадим проект **con123**. Наберем код программы в теле метода **Main**:

```
Console.Write("Введите начальную скорость Vo (от 12 до 40) м/с :  ");
double Vo = double.Parse(Console.ReadLine());          // ввод и преобразование
double t = 0, y = 0, V = Vo, g = 9.81 ;          // задание начальных условий
       //  расчет максимальной высоты подъема тела в цикле с предусловием
while (V >= 0)
 {
   V = Vo - g*t;  y = Vo*t - g*t*t/2;  t += 0.001;
 }
Console.WriteLine("время = {0:f2} с,  макс высота = {1:f2} м", t, y);
```

```
// расчет времени  полета тела в цикле с постусловием
do {
     t += 0.001;  y = Vo*t - g*t*t/2;
   }
while (y > 0);
Console.WriteLine("время полета = {0:f2} c", t);
Console.ReadKey();
```

2. Протестируем программу. Откорректируем программный код.

Многие задачи требуют многократного ввода данных. Для этого можно использовать бесконечный цикл **while(true)**, выход из которого выполняется в результате проверки вводимых данных оператором **if** в теле цикла, или цикл **do…while**() с проверкой в его постусловии. Сравним эти способы на примере ввода и суммирования чисел. Для завершения вводится буква «Q».

Пример 4

*Реализация многократного ввода: бесконечный цикл **while**.*

1. Создадим проект **con124**. Наберем код программы в теле метода **Main**:

```
    // объявляем и инициализируем переменные
double n = 0, sum = 0; string s = "";
while (true)                                    // бесконечный цикл while
 {
  Console.Write("Введите число: ");
  s = Console.ReadLine();                       // ввод
  if (s == "Q") break;                          // выход из цикла по условию
  if (double.TryParse(s, out n))                // преобразование в double
    { sum += n;                                 // суммирование чисел
      Console.WriteLine("сумма = " + sum);  }
 }
```

2. Протестируем программу. Откорректируем программный код.

Пример 5

*Реализация многократного ввода: цикл **do…while**.*

1. Создадим проект **con125**. Наберем код программы в теле метода **Main**:

```
            // объявляем и инициализируем переменные
double n = 0, sum = 0; string s = ""; do
 {
   Console.Write("Введите число: ");
   s = Console.ReadLine();                      // ввод
   if (double.TryParse(s, out n))               // преобразование в double
         {  sum += n;                                 // суммирование чисел
       Console.WriteLine("сумма = " + sum);    }
  // повторяем ввод до тех пор, пока не введен символ «Q»
  } while (s!="Q");                             // выход из цикла по условию
```

2. Протестируем программу. Откорректируем программный код.

Пример 6

*Вклады в банках с простыми и сложными процентами (цикл **while**).*

1. Создадим проект **con126**. Наберем код программы в теле метода **Main**:

```
Console.Write("Введите начальный вклад (от 100 до 500 руб: ");
double vklad = double.Parse(Console.ReadLine());
Console.Write("Введите процентную ставку (от 10 до 30): "); double
proc = double.Parse(Console.ReadLine());
        // задаем начальные значения
double sum1=vklad, sum2=vklad;  int god=0;
while (sum1 < 2*vklad)                                  // условие - удвоения суммы
 {     // выполняем вычисления накопленных сумм и задаем приращение года sum1
  += proc*vklad/100; sum2 *= (1+proc/100); god++; Console.WriteLine("год {0},
  банк1 = {1}, банк2 = {2}", god, sum1, sum2);
 }
Console.ReadKey();
```

Протестируем программу. Откорректируем программный код.

Порядок выполнения работы

Создайте консольные приложения, в которых выполняются заданные действия:

Рабочее задание

1. Вычисляется сумма всех нечетных чисел от ***n*1** до ***n*2** (ввод ***n*1**, ***n*2**, цикл for).

2. Вычисляется сумма квадратов **n** натуральных чисел, начиная с **k** (ввод **k** и **n**, цикл for).

3. Повторяются вычисления площади круга по вводимому радиусу **r** до тех пор, пока не введена буква **z** или **Z**.

4. Генерируется 8 случайных чисел в интервале (–30, 30). Выводятся эти числа и сообщения: отрицательное – положительное, четное – нечетное (for, if).

5. Генерируется 10 случайных чисел в интервале (– 20, 20). Выводятся толь- ко положительные числа и сообщения: четное – нечетное (for, if).

6. Генерируются случайные числа в интервале (–40, 40) до тех пор, пока очередное число не превышает 30. Выводятся только нечетные числа и сооб- щения: отрицательное – положительное (while, if).

7. Генерируются случайные числа в интервале (0, 20) до тех пор, пока их сумма не превысит S (вводится с клавиатуры). Нумеруются и выводятся эти числа и их сумма (цикл while).

8. Ежедневно количество бактерий увеличивается на *p* %. Через сколько дней количество бактерий увеличится в n раз (ввод *p* и *n*).

Контрольные вопросы

1. Как реализовать цикл for в C#?
2. Как реализовать цикл while в C#?
3. Как реализовать цикл do-while в C#?
4. Как выйти из цикла раньше с помощью ключевого слова break в C#?
5. Как пропустить текущую итерацию цикла и перейти к следующей с помощью ключевого слова continue в C#?
6. Как реализовать вложенные циклы (цикл внутри цикла) в C#?
7. Как реализовать бесконечный цикл с помощью ключевого слова while(true) в C#?
8. Как реализовать цикл foreach для перебора элементов в коллекции (например, массиве) в C#?
9. Как выйти из внешнего цикла из вложенного цикла с помощью ключевого слова break в C#?
10. Как организовать циклическую логику с использованием рекурсии в C#?

Практическая работа №9
Работа с массивами

Цель работы: формирование навыков работы с одномерными массивами в C#.

Теоретическая часть

Массив – именованная упорядоченная последовательность однотипных элементов, к каждому из которых можно обратиться по индексу. Для обращения к элементу массива после его имени указывается индекс элемента в квадратных скобках: w[4], z[i]. Во многих случаях индекс можно считать порядковым номе- ром элемента. В C# начальный индекс = 0 (элементы нумеруются с нуля).

Виды массивов в C#: одномерные, многомерные (например, двумерные, прямоугольные), массивы массивов (jagged – используются термины: ступенча- тые, изрезанные и др.).

Массив относится к ссылочным типам данных (располагается в хипе), по- этому создание массива начинают с выделения памяти под его элементы. Эле- ментами массива могут быть величины как значимых, так и ссылочных типов (в том числе массивы), например:

```
int[] w = new  int[10];                    // массив из 10 целых чисел;
string[] z = new  string[100];             // массив из 100 строк;
Car[] s = new  Car[5];                     // массив из 5 объектов типа Car;
double[,]  tb = new double[2, 10];   // прямоугольный массив 2 × 10; int[][] a =
new int[2][];                              // массив массивов.
```

Элементы массивов значимых типов хранят значения, массивы ссылочных типов – ссылки на элементы. При объявлении массива его элементам присваи- ваются значения по умолчанию: 0 для значимых типов и *null* для ссылочных.

Размерность массива (количество элементов в массиве) задается при объ- явлении (выделении памяти) и **не может** быть изменена впоследствии.

Размерность может задаваться числом или выражением, например:

```
int n = 5;  string[] z = new string[2*n + 1].
```

Варианты и примеры описания одномерного массива:

```
int[] a;  // объявлена только ссылка на массив, память под элементы не выделена;
int[] b = new int[4];                      // 4 элемента, значения равны 0;
int[] c = { 61, 2, 5, -9 };                // размерность вычисляется;
int[] e = new int[4] { 61, 2, 5, -9 };     // избыточное описание.
```

С элементами массива можно делать все, что допустимо для переменных того же типа. При работе с массивом автоматически выполняется контроль вы- хода за его границы: если значение индекса выходит за границы массива, гене- рируется исключение IndexOutOfRangeException (см. работу 1.6).

Наиболее распространенные задачи работы с массивами:

– поиск индекса и значений заданного элемента (например, первого, последнего, положительного, отрицательного, максимального и т. п.);

– определение количества, суммы, произведения, среднего арифметическо- го заданных элементов (например, положительных, отрицательных, от 2 до 8);

– сортировка и анализ возможных вариантов расположения элементов. При работе с массивами используются циклы. Многие действия с массивами упрощаются при использовании алгоритмической конструкции перебора элементов массива:

foreach (*тип перем* **in** *имя_массива*)
{ тело_цикла }

где *перем* – имя локальной переменной цикла, которая по очереди принимает все значения элементов массива.

Например для массива:

```
int[] mas =  { 24, 50, 18, 3, 16, –7, 9, –1 };
foreach ( int x  in  mas )          Console.WriteLine( x );
```

будут последовательно выведены все числа.

Все массивы в C# имеют общий базовый класс **Array**, определенный в про- странстве имен **System**. Некоторые свойства и методы класса **Array**:

– **Length** (свойство) – количество элементов массива (по всем размер- ностям);

– **IndexOf (LastIndexOf)** (статический метод) – поиск первого (последнего) вхождения элемента в одномерный массив;

– **Sort** (статический метод) – упорядочивание элементов одномерного массива;

– **Reverse** – изменение порядка следования элементов на обратный.

Пример 1

Формирование и вывод массивов чисел.

1. Создадим проект **con131**. Наберем код программы в теле метода **Main**:

```
Console.Write("Введите размерность массивов (от 5 до 20) "); int n =
int.Parse(Console.ReadLine());
int[] a = new int[n];  int[] b = new int[n];          // объявление массивов
for (int i = 0; i < n; i++)
 {
   a[i] = i; b[i] = a[i]*a[i];                        // заполнение массивов
   Console.WriteLine("a[{0}] = {1}, b[{0}] = {2}", i, a[i], b[i] );
 }
Console.ReadKey();
```

2. Протестируем программу. Откорректируем программный код.

Пример 2

Формирование и вывод массива строк.

1. Создадим проект **con132**. Наберем код программы в теле метода **Main**:

```
      // объявляем и заполняем массив дней недели
string[] dw = {"Вс","Пн","Вт","Ср","Чт","Пт","Сб"};
      // выводим рабочие дни с помощью цикла for
for (int i = 1; i < dw.Length - 1; i++) Console.WriteLine(i +
   " рабочий день " + dw[i]);
      // перебираем и выводим все элементы с помощью цикла foreach
foreach (string day in dw)
          Console.WriteLine(day);
     // сначала только объявляем массив месяцев, затем заполняем два элемента
string[] ms = new string[12]; ms[6] =
"Июль"; ms[7] = "Август"; int j = 0;
      // перебираем и выводим элементы с помощью цикла foreach
foreach (string m in ms) Console.Write("{0}-{1}
     ", ++j, m);
Console.ReadKey();
```

2. Протестируем программу. Проанализируем результат.

Пример 3

*Свойства и методы класса **Array**.*

1. Создадим проект **con133**. Наберем код программы в теле метода **Main**:

```
Console.Write("Введите размерность массивов (от 5 до 20) "); int n =
int.Parse(Console.ReadLine());
Random r = new Random();
                                        // создание экземпляра класса Random
```

```
int[] a = new int[n];                                   // объявление массива
for (int i = 0; i < n; i++)
  {
   a[i] = r.Next(-20, 20);          //заполнение массива случайными числами
   Console.Write("{0,4}", a[i]);                        // вывод исходного массива
  }
    // подсчет суммы и количества отрицательных чисел
   int sum = 0, num = 0;                                // задание начальных значений
foreach (int x in a)
  if (x < 0)  {  sum += x; num++;  }
Console.WriteLine("\n Сумма отрицательных = {0}, к-во = {1}", sum, num);
int max = a[0];                                         // поиск максимального элемента
foreach (int x in a) if (x > max) max = x;
Console.WriteLine("max = " + max);
Array.Sort(a);                                          // сортировка элементов массива
foreach (int x in a) Console.Write("{0,4}", x);
Console.WriteLine();
Array.Reverse(a);                                       // изменение порядка следования
foreach (int x in a) Console.Write("{0,4}", x);
Console.ReadKey();
```

2. Протестируем программу. Сравним вывод элементов двумя способами.

Порядок выполнения работы

Создайте консольные приложения, в которых выполняются заданные действия:

Рабочее задание

1. По введенному порядковому номеру выводится название дня недели и количество дней до Вс.

2. По введенному обозначению транспортного средства (a – автомобиль, v – велосипед, m – мотоцикл, p – поезд, s – самолет) выводится его название и средняя скорость.

3. Заданы диагонали мониторов в дюймах: 10.1; 11.6; 14; 15.6; 17; 19; 24; 27. Формируется и выводится таблица перевода диагоналей в сантиметры.

4. Формируется массив из *n* натуральных нечетных чисел. Выводятся чис- ла кратные 3.

5. Формируется массив из *n* целых случайных чисел от 10 до 99. Выводят- ся четные числа и их количество.

6. Формируется массив из *n* целых случайных чисел от –40 до 40. Выво- дятся нечетные отрицательные числа и их количество.

7. Формируется массив из *n* целых случайных чисел от –50 до 50. Массив упорядочивается: а) выводится сумма и количество положительных чисел; б) Выводятся числа от –20 до +20.

8. *По введенному порядковому номеру месяца и дате выводится: название месяца, количество дней в нем, количество дней, оставшихся до конца текуще- го месяца, название следующего месяца.

Контрольные вопросы

1. Как объявить и инициализировать одномерный массив в C#?
2. Как получить длину (количество элементов) одномерного массива в C#?
3. Как получить значение конкретного элемента в одномерном массиве по указанному индексу в C#?
4. Как изменить значение конкретного элемента в одномерном массиве по указанному индексу в C#?

5. Как перебрать все элементы одномерного массива с помощью цикла в C#?
6. Как найти минимальное или максимальное значение в одномерном массиве в C#?
7. Как отсортировать одномерный массив в C#?
8. Как проверить наличие определенного значения в одномерном массиве в C#?
9. Как скопировать одномерный массив в другой массив в C#?
10. Как преобразовать одномерный массив в список (List) в C#?

Практическая работа №10
Работа с двумерными массивами

Цель работы: формирование навыков работы с двумерными массивами в C#.

Теоретическая часть

В языке C# различают двумерные массивы двух видов: прямоугольные (таблица с одинаковым количеством элементов в строках) и ступенчатые. При- меры описания **прямоугольных** двумерных массивов:

```
int[,] a;                  // объявлена только ссылка, память под элементы не выделена;
int[,] b = new int[2, 3];          // 2 строки, 3 столбца, элементы равны 0 int[,] c =
{ {1, 2, 3}, {4, 5, 6} };          // размерность вычисляется
int[,] d = new int[2,3] { {1, 2, 3}, {4, 5, 6} };  // избыточное описание.
```

К элементу двумерного массива обращаются, указывая номер строки и столбца, на пересечении которых он расположен **b[i,j]**. Первый индекс всегда воспринимается компилятором, как номер строки.

Пример 1

Формирование и вывод двумерного массива заданных чисел.

1. Создадим проект **con141**. Наберем код программы в теле метода **Main**:

```
    // формируем прямоугольный массив чисел из двух строк
int[,] ar = { { 11,12,13,14,15 }, { 21,22,23,24,25 } };
                    // с помощью foreach все элементы выводятся в одну строку
foreach (int x in ar) Console.Write("{0,4}", x);
   Console.WriteLine();
      // выводим числа построчно (в форме таблицы)
for (int i = 0; i < 2; i++)
  {
     for (int j = 0; j < 5; j++)  Console.Write("{0,4}", ar[i,j]);  // j-я строка
    Console.WriteLine();                    // переход на следующую строку
  }
Console.ReadKey();
```

2. Протестируем программу. Откорректируем программный код.

Пример 2

Формирование и вывод таблицы чисел.

1. Создадим проект **con142**. Наберем код программы в теле метода **Main**:

```
Console.Write("Введите количество строк (n<9):  "); int n =
int.Parse(Console.ReadLine()); Console.Write("Введите
```

```
количество столбцов (m<9):                          "); int m =
int.Parse(Console.ReadLine());
        // объявляем массив чисел из n строк и m столбцов
int[,] mas = new int[n, m];
        // построчно заполняем массив целыми числами
for (int i = 0; i < n; i++)
 {
   for (int j = 0; j < m; j++) mas[i,j] = 10*i + j;            // i-я строка
 }
        // построчно выводим элементы массива (в форме таблицы)
for (int i = 0; i < n; i++)
 {
  for (int j = 0; j < m; j++)
     Console.Write("{0,4}", mas[i,j]);          // вывод элементов i-й строки
  Console.WriteLine();                          // переход на следующую строку
 }
  Console.ReadKey();
```

2. Протестируем программу. Откорректируем программный код.

В **ступенчатых** массивах количество элементов в разных строках может различаться. В памяти хранится ступенчатый массив иначе, чем прямоугольный – в виде нескольких внутренних массивов, каждый из которых имеет свой размер (массив массивов). Кроме того, выделяется отдельная область памяти для хра- нения ссылок на каждый из внутренних массивов. Под каждый из массивов, со- ставляющих ступенчатый массив, память требуется выделять явным образом.

Примеры описания ступенчатых двумерных массивов:

`int[][] a = new int[3][];` // выделение памяти под ссылки на 3 строки `a[0] = new int[5];` // выделение памяти под 0-ю строку (5 элементов) `a[1] = new int[3];` // выделение памяти под 1-ю строку (3 элемента) `a[2] = new int[4];` // выделение памяти под 2-ю строку (4 элемента)

Сокращенный вариант: int[][] a = { new int[5], new int[3], new int[4] };

Обращение к элементам ступенчатого массива: a[0][3], a[1][2], a[i][j].

Пример 3

Формирование и вывод ступенчатого массива чисел.

1. Создадим проект **con143**. Наберем код программы в теле метода **Main**:

```
// построчно объявляем  и заполняем массив из 3-х одномерных массивов
int[][] jag = new int[3][]
 {  new int[] {3,7,9,5,12},   new int[] {2,4},  new int[] {1,3,5}  };
                    // построчно выводим три внутренних одномерных массива
foreach (int[] arr in jag)
  { foreach (int a in arr) Console.Write("{0,4}", a);
   Console.WriteLine();
  }
  Console.ReadKey();
```

2. Протестируем программу. Откорректируем программный код.

Порядок выполнения работы

Создайте консольные приложения, в которых выполняются заданные действия:

Рабочее задание

1. Формируется и выводится прямоугольный массив (5 строк и 6 столбцов) целых случайных чисел от –40 до 40. Вычисляется и выводится: а) сумма чисел в каждой строке; б) среднее арифметическое чисел в каждой строке;

2. Формируется и выводится прямоугольный массив (n строк и m столб- цов)

целых случайных чисел от –50 до 50. Вычисляется и выводится: а) среднее арифметическое отрицательных чисел в каждой строке; в) сумма и среднее арифметическое положительных четных чисел в каждой строке; д) сумма и среднее арифметическое всех чисел.

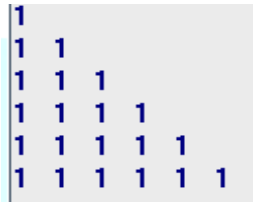
3. Формируется и выводится прямоугольный массив – таблица умножения двух чисел от 1 до 10.

4. Формируется и выводится прямоугольный массив (6 строк и 8 столбцов) целых случайных чисел от –70 до 70. Вычисляется и выводится: а) максималь- ный элемент в кажой строке; б) минимальный положительный элемент в каж- дой строке.

5. Формируется и выводится прямоугольный массив (*n* строк и *m* столб- цов) целых случайных чисел от –90 до 90. Вычисляется и выводится: а) мини- мальный из всех отрицательных элементов; б) максимальный из модулей всех элементов массива.

6. *Формируется и выводится прямоугольный массив (*n* строк и *m* столб- цов) целых случайных чисел от –80 до 80. Вычисляется и выводится: а) в каж- дой строке находится минимальный элемент и заменяется нулем; б) в каждом столбце находится максимальный элемент и заменяется единицей.

7. Формируется массив и выводится треугольная таблица, заполненная: а) единицами; б) нулями.



Контрольные вопросы

1. Как объявить и инициализировать двумерный массив в C#?
2. Как получить количество строк и столбцов в двумерном массиве в C#?
3. Как получить значение конкретного элемента в двумерном массиве по указанным индексам в C#?
4. Как изменить значение конкретного элемента в двумерном массиве по указанным индексам в C#?
5. Как перебрать все элементы двумерного массива с помощью вложенных циклов в C#?
6. Как скопировать один двумерный массив в другой в C#?
7. Как проверить наличие определенного значения в двумерном массиве и получить его индексы в C#?
8. Как отсортировать строки или столбцы в двумерном массиве в C#?
9. Как преобразовать двумерный массив в одномерный в C#?
10. Как создать "рваный" (jagged) массив в C# и как им пользоваться?

Практическая работа №11
Обработка исключений

Цель работы: формирование навыков обработки исключений.

Теоретическая часть

Важным показателем эффективности компьютерных программ является защита от сбоев. Для повышения надежности необходимо предусмотреть обра- ботку критических ситуаций, которые могут вызывать ошибки в работе прило- жений. Перечислим основные способы защиты от сбоев.

– Контроль формата вводимых данных – «умный» парсинг (метод **TryParse**): **int a; if (!int.TryParse(Console.ReadLine(), out a)) Console.WriteLine("Неверный формат"); Console.WriteLine(a);**

– Контроль допустимых значений вводимых данных, например, при инкапсуляции полей с помощью «умных» свойств (**set – get**).

– Контроль операций и результатов, например, путем локальной проверки использования встроенных методов с помощью условий, например:

if (a < 0) Console.WriteLine("Неверное значение < 0");

– Контроль критических (часто непредвиденных) ситуаций. Исключительная ситуация, или **исключение** – это возникновение события,

(часто непредвиденного или аварийного), которое порождается некорректным использованием команд или аппаратуры. Например: деление на ноль, обращение по несуществующему адресу памяти, попытка прочитать несуществующий файл. Использование механизма исключений позволяет логически разделить вычислительный процесс на две части – обнаружение аварийной ситуации и ее обработку. Типичный синтаксис:

try { контролируемый блок } **catch** { блок обработки исключения } **finally** { блок завершения }

В контролируемый блок **try** включаются потенциально опасные фрагмен- ты программного кода. Все функции, прямо или косвенно вызываемые из этого блока, также считаются ему принадлежащими.

В одном или нескольких блоках обработки исключений **catch** описывается, как обрабатываются ошибки различных типов:

catch (тип имя) { обработка конкретной ошибки } **catch (тип)** { обработка ошибки заданного типа } **catch** { обработка любой, часто неизвестной ошибки }

Необязательный блок завершения **finally** выполняется независимо от того, возникла ли ошибка в контролируемом блоке или нет. Собственные исключе- ния можно создавать оператором **throw.**

Механизм обработки исключений:

– функция или операция, в которой возникла ошибка, генерирует исключение;

– выполнение текущего блока прекращается, ищется соответствующий обработчик исключения и ему передается управление;

– если обработчик не найден, вызывается стандартный обработчик;

– наконец, выполняется блок finally, если он присутствует.

Для работы с исключениями предназначен класс **Exception**. Приведем не- которые стандартные исключения:

– **FormatException** – попытка передать в метод аргумент неверного формата;

– **InvalidCastException** – ошибка преобразования типа;

– **ArithmeticException** – ошибка арифметических операций или преобразований;

– **DivideByZeroException** – попытка деления на ноль;

– **OverFlowException** – переполнение выполнения арифметических операций;

– **IndexOutOfRangeException** – индекс массива выходит за границы диапазона;

– **OutOfMemoryException** – недостаточно памяти для создания нового объекта;

– **StackOverFlowException** – переполнение стека;

Пример типичной последовательности обработки исключений:

try { контролируемый блок }
catch (DivideByZeroException) { обработка деления на 0 } **catch** (
OverflowException) { обработка переполнения }
catch (IndexOutOfRangeException) { обработка неверного индекса массива } **catch**
(FormatException) { обработка неверного формата }
catch { обработка всех остальных исключений }

Некоторые важные свойства класса **Exception**:

– **Message** – текстовое описание ошибки (только для чтения);

– **Source** – имя объекта, сгенерировавшего ошибку;

– **TargetSite** – метод, сгенерировавший ошибку;

– **InnerException** – ссылка на исключение, послужившее причиной текущего.

Пример 1

Обнаружение и обработка ошибка деления на 0.

1. Создадим проект **con151**. Наберем код программы в теле метода **Main**:

```
Console.Write("Введите делимое ");
int a = int.Parse(Console.ReadLine());
Console.Write("Введите делитель "); int b =
int.Parse(Console.ReadLine());
// в проверяемый блок включим операцию деления и вывод результата
try {  Console.WriteLine(a / b);   }
    // обработка ошибки деления на 0 и вывод системного сообщения
catch (DivideByZeroException e)
    {  Console.WriteLine(e.Message); }
   // обработка неопознанной ошибки, вывод собственного сообщения catch
{ Console.WriteLine("Произошла ошибка"); } Console.ReadKey();
```

2. Протестируем программу. Сравним результаты обработки исключений при вводе различных целых чисел и 0.

3. Закомментируем первый блок **catch**. Сравним результаты в этом случае.

4. Снимем комментирование. Будем поочередно изменять тип переменных **a** и **b** на **double**: `double a = double.Parse(Console.ReadLine());`

5. Сравним результаты обработки исключений.

Пример 2

Задан строковый массив дней недели. Составим программу, которая по введенному номеру выводит название дня недели и количество дней до Вс, а также обрабатывает ошибки формата ввода и выхода индекса за границы массива.

1. Создадим проект **con152**. Наберем код программы в теле метода **Main**:

```
// создадим массив дней недели
```

```
string[] dw = { "Вс", "Пн", "Вт", "Ср", "Чт", "Пт", "Сб" };
Console.Write("Введите номер дня недели (0 - 6)   ");
   // в проверяемый блок включим ввод номера и вывод сообщения
try {
     int i = int.Parse(Console.ReadLine()); Console.WriteLine("{0},
     до Вс {1} дней", dw[i], 6 - i);
    }
   // обработка ошибки выхода индекса за границы диапазона
catch (IndexOutOfRangeException e)
    {  Console.WriteLine(e.Message);   }
   // обработка ошибки формата ввода
catch (FormatException e)
    {  Console.WriteLine(e.Message);   }
   // обработка неопознанной ошибки
catch {  Console.WriteLine("Ошибка");    }
Console.ReadKey();
```

2. Протестируем программу. Откорректируем программный код.

3. Сравним результаты обработки исключений при вводе чисел > 6 и букв.

4. Исследуем, как влияет порядок следования блоков **catch** на обработку исключений от одного блока **try.**

Пример 3

Составим программу, в которой обрабатывается ошибка переполнения.

1. Создадим проект **con153**. Наберем код программы в теле метода **Main**:

```
   // в проверяемый блок try включим вычисление и вывод;
   // проверяемое выражение заключим в checked
int a = 1000; int b = 3000; try {
     int pro = checked (a*a*b);
     Console.WriteLine("Произведение = {0}", pro);
    }
   // обработка ошибки переполнения
catch(OverflowException e)
   { Console.WriteLine(e.Message);  }
   // обработка неопознанной ошибки catch {
Console.WriteLine("Ошибка"); }
Console.ReadKey();
```

2. Протестируем программу. Откорректируем программный код.

3. Удалим **checked**. Сравним результат в этом случае.

Порядок выполнения работы

Создайте консольные приложения, в которых выполняются заданные дей- ствия и обрабатываются исключения:

Рабочее задание

1. Вычисляется сила тока **i = u/r** по введенным значениям напряжения и сопротивления (тип **int**):

2. Расстояния до звезд **a** = 5 и **b** = 8000 световых лет вычисляется в **км** и **м**
sa = a*365*24*3600*300 000 (км).

3. Проверяется правильность ввода логина и пароля, состоящих только из цифр.

4. Задан строковый массив месяцев. По введенному порядковому номеру месяца выводится название месяца и количество дней в нем.

5. Вычисляются значения функций от целочисленных аргументов:

а) y = (x + 5) / (x - 7) *ввод* **x**

в) z = (x – y) / cos(x) *ввод* **x** и **y**

6. Задан строковый массив городов: Брест, Витебск, Гомель, Минск, Полоцк. По введенной первой букве или номеру выводится название города.

Контрольные вопросы

1. Что такое исключение в C# и зачем оно используется?
2. Как обработать исключение с помощью блока try-catch в C#?
3. Что такое блок finally и как он связан с обработкой исключений в C#?
4. Как создать собственное пользовательское исключение в C#?
5. Как выбросить исключение в C# с использованием ключевого слова throw?
6. Как можно обработать несколько разных типов исключений в одном блоке catch в C#?
7. Как использовать блок try-finally для выполнения кода независимо от возникновения исключений в C#?
8. Как получить информацию о возникшем исключении (тип, сообщение, стек вызовов) в C#?
9. Как использовать блок using в C# для автоматического освобождения ресурсов при возникновении исключений?
10. Как перехватить и обработать необработанное исключение в C#?

Практическая работа №12
Работа с символами

Цель работы: формирование навыков работы с символами в C#.

Теоретическая часть

Для работы с символами предназначен тип **char.** Ему соответствует базо- вый класс **System.Char**. В памяти переменная типа **char** представляется число- вым кодом символа. Требуется преобразование в символ:

```
char ch = 'A';                          // в переменной ch код 65
int k = 100;    c = (char)k;            // преобразование в символ 'd' с кодом 100
```

При вводе символа с клавиатуры методом **Read()** и нажатии клавиши Enter в буфер попадает его код, а также коды нажатия клавиши (13) и конца строки (10). `int k = Console.Read();` // в переменной код символа +13+10

```
char ch = (char) Console.Read();  // в переменной символ +10+13
```

Лучше использовать метод **ReadLine()** и разборку строки:

```
char s = char.Parse(Console.ReadLine());  // в переменной только символ
```

Некоторые методы класса **Char**:

– **IsDigit(s), IsLetter(s), IsLower(s), IsUpper(s), IsPunctuation(s), IsCon- trol(s)** возвращают **true**, если символ **s** является соответственно: цифрой, бук- вой, строчной, заглавной буквой, знаком пунктуации, управляющим символом;

– **GetNumericValue(s)** возвращает числовое значение, если символ **s** цифра;

– **ToLower(s), ToUpper(s)** изменяют регистр букв (в строчные или заглавные);

– **ToCharArray(str)** преобразуют строку в массив символов.

Пример 1

Определение категории введенного символа.

1. Создадим проект **con161**. Наберем код программы в теле метода **Main**: `int k;`

```
char ch; string mes = "";                    //объявление переменных do {
    // повторение ввода в цикле с постусловием
  Console.WriteLine("Введите символ: ");
  k = Console.Read(); ch = (char)k;     // ввод и преобразование кода в символ
                                        // проверка категории символа
```

```
  if (char.IsLetter(ch)) mes = "буква"; else if
  (char.IsDigit(ch)) mes = "цифра";
  else if (char.IsPunctuation(ch)) mes = "знак пунктуации"; else
  mes = "управляющий символ";
  Console.WriteLine("Введен символ {0}, его код {1}, это {2}", ch, k, mes);
   } while (ch != 'Q');              // завершение цикла при вводе буквы Q
Console.ReadKey();
```

2. Протестируем программу. Откорректируем программный код.

Пример 2

Подсчет количества и суммы всех содержащихся в строке цифр.

Заметим, что символам цифр от 0 до 9 соответствуют коды от 48 до 57.

1. Создадим проект **con162**. Наберем код программы в теле метода **Main**:

```
Console.WriteLine("Введите строку:          ");
string str = Console.ReadLine();                    // ввод строки
int k = 0, sum = 0;                                 // задание начальных значений
foreach (char s in str)                             // перебор всех символов строки
  {
   if (char.IsDigit(s))                             //если очередной символ цифра,
    {  k++;  sum += s-48;                           //считаем к-во и сумму цифр
       Console.Write(s +" ");                       //выводим цифры
    }
  }
Console.WriteLine("\nВ строке {0} цифр, их сумма = {1}", k, sum);
Console.ReadKey();
```

2. Протестируем программу. Откорректируем программный код.

Пример 3

Перестановка соседних символов строки.

1. Создадим проект **con163**. Наберем код программы в теле метода **Main**:

```
Console.WriteLine("Введите строку: ");
string str = Console.ReadLine();
int k = str.Length-1;                       // определение  длины строки
string str2 = "";                           // объявление новой пустой строки
          // сформируем строку str2  с переставленными соседними символами
for (int i = 0; i < k ; i += 2)
  { str2 += str[i+1];    str2 += str[i]; }
         // добавим финальный нечетный символ (с четным индексом)
if (k % 2 == 0) str2 += str[k];
 Console.WriteLine(str2);                   //вывод строки str2
 Console.ReadKey();
```

2. Протестируем программу. Откорректируем программный код.

Пример 4

*Работа с массивом символов. Использование методов класса **Array.***

1. Создадим проект **con164**. Наберем код программы в теле метода **Main**:

```
Console.WriteLine("Введите строку: ");
string str = Console.ReadLine();                    // ввод строки
         // преобразование строки в массив символов
char[] ch = str.ToCharArray();
```

```
Array.Reverse(ch);   Console.WriteLine(ch);        // реверсирование и вывод
Array.Sort(ch);      Console.WriteLine(ch);        // сортировка и вывод
Console.ReadKey();
```

2. Протестируем программу. Откорректируем программный код.

Порядок выполнения работы

Создайте консольные приложения, в которых выполняются заданные дейст- вия над символами введенных строк:

Рабочее задание

1. Подсчитывается количество строчных, заглавных букв, знаков препинания в строке.
2. Подсчитывается количество вхождений заданной буквы (цифры).
3. Заданные строчные латинские буквы преобразуются в заглавные.
4. Проверяется, имеются ли в строке рядом стоящие удвоенные символы.
5. Выводятся все повторяющиеся в строке буквы и подсчитывается количе- ство каждой из них.
6. Подсчитывается количество и сумма всех содержащихся в строке четных (нечетных) цифр.
7. Первые строчные буквы слов строки преобразуются в заглавные.
8. *Содержащиеся в строке цифры 0–5 заменяются на слова «ноль»…«пять».

Контрольные вопросы

Работа со строками

Цель работы: формирование навыков работы со строками.

Теоретическая часть

Для работы со строками предназначены классы **String** и **StringBuilder.** Строки типа **string** в **C#** – **неизменяемый** тип данных. Методы, изменяющие содержимое строки, на самом деле создают новую копию строки, а неиспользу- емые «старые» копии автоматически удаляются сборщиком мусора.

Примеры создания строк:

```
string s;                    // переменная объявлена, инициализация отложена;
string st = "строка";              // инициализация строковым литералом; string
u = new string(“ “, 20);   // создание с помощью конструктора; char[] a = { 'e',
'n',                        'd' };  // создание массива символов;
string v = new string( a );        // строка из массива символов.
```

Основные операции для работы со строками: присваивание **=** ; проверка на равенство **= =;** на неравенство **!=;** сцепление (конкатенация) строк **+;** обращение к символу строки по индексу **[k].**

Строки равны, если имеют одинаковое количество символов и совпадают посимвольно. Обращаться к отдельному символу строки по индексу можно только для получения значения, но не для его изменения.

Длина строки (количество символов) определяется свойством **Length.** Пу- стая строка – экземпляр класса **String**, содержащий **0** символов: **string s = "".**

Приведем некоторые методы класса **System.String:**

– **Compare, CompareTo** – сравнение строк;

– **Concat** – конкатенация (сцепление) строк;

– **Copy** – создание копии строки;

– **IndexOf(s), LastIndexOf(s)** – определение позиции первого (последнего) вхождения подстроки или символа s;

– **Substring(k1, k2)** – выделение подстроки с позиции k1 по k2;

– **Replace(s,z)** – замена всех вхождений подстроки **s** новой подстрокой или символом **z**;

– **Insert(k)** – вставка подстроки с позиции k;

– **Remove (k,n)** – удаление n символов с позиции k;

– **Trim(), TrimStart(),TrimEnd()** – удаление концевых пробелов;

– **Split(d)** – разделение строки в массив строк по символу-разделителю d (или массиву символов);

– **Join(d, mas)** – слияние массива строк mas в единую строку с разде- лителем d;

– **Format** – форматирование с заданными спецификаторами формата.

Некоторые спецификаторы формата строк:

– **C** или **c** вывод значений в денежном (currency) формате;

– **F** или **f** вывод значений с фиксированной точностью;

– **G** или **g** формат общего вида;

– **P** или **p** вывод числа в процентном формате.

Пример 1

*Использование методов класса **String**.*

1. Создадим проект **con171**. Наберем код программы в теле метода **Main**:

```
        //  исходная строка заимствована из пособии [1]
string str = "прекрасная королева"; Console.WriteLine(str);
                                                    // вывод
        //выделяем подстроку str2, удаляем символы  “ле”
string str2 = str.Substring(3).Remove(12, 2);
  Console.WriteLine(str2);                          // вывод:  красная корова
        // расщепляем строку str в массив слов mas (разделитель – пробел)
string[] mas = str.Split(' ');
        // выводим все слова s массива mas
  foreach (string s in mas) Console.WriteLine(s);
   Array.Sort(mas);              // сортируем слова массива mas по алфавиту
  foreach (string s in mas) Console.WriteLine(s);         // вывод
        // соединяем слова массива mas в одну строку str3
string str3 = string.Join(" !!! моя ", mas);
         Console.WriteLine(str3);           // вывод: королева !!! моя прекрасная
       string str4 = str3.Replace("!", "?");                  // заменяем все ! на ?
        Console.WriteLine(str4);           // вывод: королева ??? моя прекрасная
        // определяем позицию k вхождения подстроки моя
int k = str4.IndexOf("моя");
                    //удаляем из строки str4  символы начиная с k-го до конца
string str5 = str4.Remove(k);
  Console.WriteLine(str5);                          // вывод: королева ???
Console.ReadKey();
```

2. Протестируем программу. Откорректируем программный код.

Для создания изменяемых строк предназначен класс **StringBuilder**, кото- рый определен в пространстве имен **System.Text**. Требует создания экземпляра! Позволяет **изменять** значение своих экземпляров. При создании экземпляра обязательно использовать **new** и конструктор, например:

– StringBuilder a = new StringBuilder();

– StringBuilder b = new StringBuilder("Privet");

– StringBuilder d = new StringBuilder("Privet", 10).

Некоторые методы класса **StringBuilder:**

– **Append(s)** – добавление в конец строки;

– **AppendFormat(…)** – добавление форматированной строки;
– **Insert(k)** – вставка подстроки с позиции **k**;
– **Remove (k, n)** – удаление **n** символов с позиции **k**;
– Replace(s, z) – замена всех вхождений подстроки **s** новой подстрокой или символом **z;**
– **ToString()** – преобразование в строку типа **string;**.
– **Capacity** – получение или установка емкости буфера. Если устанавлива- емое значение меньше текущей длины строки или больше максимального, гене- рируется исключение ArgumentOutOfRangeException;
– **MaxCapacity** – **м**аксимальный размер буфера.

Пример 2

Использование методов класса StringBuilder.

1. Создадим проект **con172**. Наберем код программы в теле метода **Main**:

```
using System;
   using System.Text;                    // подключение пространства имен System.Text
class Program
{
 static void Main()
 {
  Console.Write("Введите зарплату: ");
  double zar = double.Parse(Console.ReadLine());
  StringBuilder str = new StringBuilder();          // создание объекта
  str.Append("зарплата ");
                    // добавляем строку в денежном формате  (тыс и млн отделяются)
  str.AppendFormat("{0,6:C} - за год {1,6:C}", zar, zar*12);
    Console.WriteLine(str);                                  // вывод
  str.Replace("p.", "$.");                                   // замена p. на $
    Console.WriteLine("А было бы лучше: " + str);            // вывод
  Console.ReadKey();
 }
}
```

2. Протестируем и откорректируем программу.

Порядок выполнения работы

Создайте консольные приложения, в которых выполняются заданные дей- ствия над введенными строками:

Рабочее задание

1. Все пробелы в строке заменяются на символы подчеркивания _.
2. Стоящие рядом две точки заменяются на три звездочки ***.
3. Из строки удаляется заданное слово.
4. Во введенной строке заданное слово заменяется на другое.
5. Выводится подстрока, расположенная до последней запятой.
6. Выводится подстрока, расположенная после первого двоеточия.
7. Выводится первое и последнее слова строки.
8. Подсчитывается количество слов в строке.

9. Подсчитывается количество слов в строке, начинающихся с заданной буквы.

10. Меняется местами соседние слова.

11. Выводятся слова, заключенные в кавычки « » (в угловые скобки < >).

12. Выводятся все слова, начинающиеся с заданной буквы.

Контрольные вопросы

1. Как объявить и инициализировать строковую переменную в C#?
2. Как получить длину строки в C#?
3. Как объединить две строки в C#?
4. Как разделить строку на подстроки с помощью разделителя в C#?
5. Как проверить, содержит ли строка определенную подстроку в C#?
6. Как заменить все вхождения определенной подстроки в строке на другую строку в C#?
7. Как преобразовать строку в верхний или нижний регистр в C#?
8. Как очистить лишние пробелы в начале и конце строки в C#?
9. Как проверить, является ли строка числом (числовым значением) в C#?
10. Как преобразовать числовое значение в строку в C#?

Практическая работа №13
Использование регулярных выражений

Цель работы: формирование навыков использования регулярных выраже- ний в C#.

Теоретическая часть

Регулярное выражение (*regular expression*) – шаблон, по которому выполняется поиск соответствующего ему фрагмента текста.

Регулярные выражения предназначены для поиска в тексте фрагментов (символов, строк) по заданному шаблону с целью дальнейшей обработки.

Язык описания регулярных выражений содержит символы двух видов: обычные и метасимволы. Обычный символ представляет в шаблоне сам себя. Метасимвол представляет: класс символов (например, **\d** обозначает цифру), уточняющий символ или квантификатор (например, **A{3}** означает, что букву **A** необходимо повторить три раза). Чтобы одиночный метасимвол в шаблоне представлял сам себя, его необходимо экранировать обратным слешем **“\”**, а целое выражение – символом **@,** например: \\ воспринимается как один слеш.

Приведем примеры часто используемых метасимволов. Метасимволы, представляющие класс символов:

. (точка) любой символ;

[ ] любой *одиночный* символ из набора внутри квадратных скобок;

[^] любой *одиночный* символ, не входящий в набор внутри скобок;

\w любой алфавитно-цифровой символ, то есть буква или десятичная цифра;

\W любой **не** алфавитно-цифровой символ, то есть кроме букв и цифр;

\s любой пробельный символ, то есть пробел, табуляция (\t, \v), перевод строки (\n, \r), новая страница (\f);

\S любой **не** пробельный символ;

\d любая десятичная цифра;

\D любой символ, не являющийся десятичной цифрой. Уточняющие метасимволы

^ искать только с начала строки, **$** искать с конца строки;

\b встречается только в начале или конце слова; **\B** только внутри слова.

Квантификаторы (задают количество повторений предыдущего элемента):

***** 0 или более повторений; **?** 0 или одно; **+** одно или более повторений;

{n} ровно **n** повторений; **{n,m}** от **n** до **m** повторений.

При задании квантификаторов для последовательности применяется **груп- пирование** с помощью круглых скобок.

Примеры простых шаблонов для поиска: семизначного номера телефона вида NNN-NN-NN:

[0-9]{3}-[0-9]{2}-[0-9]{2} или **\d{3}-\d{2}-\d{2} ;**

номера автомобиля (4 цифры, 2 буквы, дефис, цифра от 1 до 7):

\d{4}[ABEIKMHOPCTX]{2}-[1-7] ;

Регулярные выражения в .NET реализуются несколькими классами прост- ранства имен **System.Text.RegularExpressions**. Основной класс **Regex.** Возмож- ны два способа обработки текста: 1) вызов статических методов класса **Regex** с передачей в качестве параметров исходной строки и шаблона; 2) создание объекта **Regex** с передачей шаблона в конструктор класса.

Некоторые методы класса **Regex.**

IsMatch(str, reg) – поиск вхождения подстроки с шаблоном **reg** в строку **str;**
Split(text) – разбиение текста **text** на массив строк по шаблону-разделителю;
Match или **Matches** – извлечение из текста одного или всех вхождений.

Метод Match возвращает одноименный объект Match с информацией о сов- падении. Метод Matches возвращает коллекцию MatchCollection, в которую вхо- дят объекты Match для всех совпадений в проанализированном тексте.

Пример 1

Проверка телефонного номера вида NNN-NN-NN.

1. Создадим проект **con181**. Наберем код программы:

```
using System;
// подключаем пространство имен System.Text.RegularExpressions
using System.Text.RegularExpressions; class
Program
{
  static void Main()
  {
    Console.WriteLine("Введите номер телефона");
    String str= Console.ReadLine();
// создаем шаблон регулярного выражения
    String reg = @"^\d{3}-\d{2}-\d{2}$";
// проверяем совпадение введенной строки с шаблоном
    if (Regex.IsMatch(str, reg))
        Console.WriteLine("{0} похоже", str);
    else Console.WriteLine("{0} ошибка", str);
    Console.ReadKey();
  }
}
```

2. Протестируем программу. Откорректируем программный код.

3. Модифицируем шаблон для определения номера телефона с разделени- ем групп цифр пробелом или дефисом

```
String reg = @"^\d{3}[ -]\d{2}[ -]\d{2}$";
```

Пример 2

Разбиение текста на слова.

1. Создадим проект **con182**. Наберем код программы в теле метода **Main**:

```
// задаем исходную строку и шаблон разделителя
string text = " салат - 4 руб, борщ - 10 руб, чай - 1 руб."; string
reg = "[-,.]+";
Regex  r = new Regex(reg);                    // создаем объект класса Regex
```

```
// разбиваем строку на слова по шаблону
string[] words = r.Split(text); foreach
( string wrd in words )
   Console.WriteLine(wrd);
Console.ReadKey();
```

2. Протестируем программу. Откорректируем программный код.

Пример 3

Замена фрагментов текста.

1. Создадим проект con183. Наберем код программы в теле метода Main:

```
// задаем исходную строку и шаблоны
string text = "телефон 400 рб, часы 120 грн, компьютер 560 $"; string
reg = "рб|грн|\\$";                                    // шаблон для поиска
string zam = "руб";                                    // шаблон для замены
// выполняем поиск и замену по шаблонам и сразу вывод
Console.WriteLine(Regex.Replace(text, reg, zam));
Console.ReadKey();
```

2. Протестируем программу. Откорректируем программный код.

Порядок выполнения работы

Создайте консольные приложения, в которых выполняются заданные дей- ствия над введенными строками:

Рабочее задание

1. Проверяется, есть ли в строке заданное слово (в любом регистре).
2. Проверяется, есть ли в строке слово из N букв.
3. Проверяется, содержит ли строка число из N цифр.
4. Проверяется, содержит ли строка цифры (знаки препинания, только буквы).
5. Все пробелы в строке заменяются на два символа подчеркивания __
6. Все знаки препинания (**. , ; : ? !**) в тексте заменяются на символ звездочка*.
7. Подсчитывается количество строчных (прописных) букв.
8. Подсчитывается количество удвоенных согласных (гласных).
9. Выводятся только слова, которые содержат заданное количество букв.
10. Выводятся слова, которые содержат не менее заданного количества букв.
11. Строка разбивается на слова, которые выводятся в обратном порядке.
12. Проверяется корректность ввода номера автомобиля.
13. Проверяется, содержит ли введенный логин не менее n букв, а пароль не менее m цифр.
14. *Проверяется корректность ввода адреса электронной почты.
15. *Проверяется корректность IP-адреса.
16. *Подсчитывается количество и сумма всех содержащихся в строке цифр.

Контрольные вопросы

1. Как использовать регулярные выражения для проверки соответствия строки определенному шаблону в C#?
2. Как найти все вхождения определенного шаблона в строке с помощью регулярных выражений в C#?
3. Как извлечь определенные части строки, соответствующие определенным шаблонам, с помощью регулярных выражений в C#?
4. Как заменить все вхождения определенного шаблона в строке с помощью регулярных выражений в C#?

5. Как использовать специальные символы и метасимволы в регулярных выражениях в C#?
6. Как использовать квантификаторы (например, *, +, ?) для указания количества повторений в регулярном выражении в C#?
7. Как использовать группировку и обратные ссылки в регулярных выражениях в C#?
8. Как использовать символьные классы (например, \d, \w, \s) для сопоставления категорий символов в регулярных выражениях в C#?
9. Как использовать альтернативы (|) для указания нескольких возможных вариантов в регулярном выражении в C#?
10. Как использовать модификаторы (например, ^, $, \b) для указания позиции в регулярном выражении в C#?

Практическая работа №14
Работа с файлами

Цель работы: формирование навыков работы с файлами.

Теоретическая часть

Обмен данными с устройствами в C# выполняется с помощью подсистемы ввода-вывода (**IO**) и классов библиотеки .NET. Реализуется с помощью потоков.

Поток (*stream*) – абстрактное понятие, относящееся к любому переносу данных от источника к приемнику и наоборот. Поток определяется как после- довательность байтов и не зависит от конкретного устройства, с которым про- изводится обмен. Потоки обеспечивают единообразие при работе со стандарт- ными типами данных и с типами, определяемыми пользователем. Обмен с по- током для повышения скорости передачи данных производится, как правило, через **буфер**, который выделяется для каждого открытого файла.

Чтение (ввод – input) – передача данных с внешнего устройства в опера- тивную память, обратный процесс – **запись** (вывод – output).

Для работы с потоками и файлами необходимо подключать пространство имен **System.IO.** Выполнять обмен с внешними устройствами можно на уровне:

– двоичного представления данных – классы **BinaryReader, BinaryWriter**;

– байтов – класс **FileStream**;

– текста (строк) – классы **StreamWriter, StreamReader**.

Доступ к файлам может быть:

– последовательным – очередной элемент можно прочитать (записать) толь- ко после предыдущего элемента;

– произвольным или прямым – выполняется чтение (запись) произвольно- го элемента по заданному адресу.

Прямой доступ при отсутствии дальнейших преобразований обеспечивает более высокую скорость получения нужной информации. В двоичных и байтовых потоках можно использовать оба метода доступа. Для текстовых файлов возмо- жен только последовательный доступ.

Для открытия файла на запись текста создается поток – объект класса **StreamWriter.** Параметрами конструктора служат имя файла **imf** и режим записи (true – дозапись, false – перезапись):

StreamWriter **fw** = new StreamWriter(**imf**, true);.

Заметим, что символы, которые можно ошибочно принять за управляющие, в имени файла необходимо экранировать, например, слешами "**D:\\work\\my.txt**", или все имя целиком с помощью символа «собака» **@"D:\work\my.txt"**.

Для дальнейшей работы используется имя (дескриптор) созданного объекта **fw** и стандартный метод вывода строки, например:

fw.WriteLine("Запись в файл");

По завершению работы поток вывода закрывается **fw.Close();**

Для открытия файла на чтение создается поток – объект класса **StreamReader:**

StreamReader **fr** = new StreamReader(**imf**);

Текст файла можно читать в строковую переменную **s** целиком (одной строкой):

string **s = fr.ReadToEnd();** или string s = **fr.ReadAllText(imf);**

а также построчно, и сразу выводить на консоль: string s; int i = 0;

while ((**s = fr.ReadLine()**) != null) Console.WriteLine("{0} : {1}", ++i, s);

Можно также читать строки файла в массив для дальнейшего вывода, например:

string[] stroki = **fr.ReadAllLines(imf)**;
foreach (string **s** in **stroki**) Console.WriteLine(**s**);

Здесь важно отметить, что при чтении-записи могут возникать критические ошибки, поэтому следует обрабатывать исключения, помещая соответствующие фрагменты кода в блок **try**, например:

try { StreamReader **frr** = new StreamReader(**imf**);
string **s = frr.ReadToEnd();** Console.WriteLine(s);
frr.Close(); **}**
catch(FileNotFoundException **ex**) **{**
Console.WriteLine(**ex**.Message); Console.WriteLine("Проверьте имя файла!"); return; **}**

Пространство имен **System.IO** содержит классы **File, FileInfo; Directory, DirectoryInfo** для работы с файлами и каталогами (папками), например: созда- ние, удаление, перемещение файлов и каталогов, получение свойств.

Приведем примеры некоторых методов:

– **Create, CreateSubDirectory** – создать каталог (подкаталог) по указанно- му пути;

– **MoveTo** – переместить каталог и все его содержимое на новый адрес;

– **Delete** – удалить каталог со всем его содержимым;

– **GetFiles** – получить файлы текущего каталога, как массив объектов **FileInfo**;

– **GetDirectories** – получить массив подкаталогов.

Пример 1

*Запись в текстовый файл **my.txt** и чтение строк.*

1. Создадим проект **con191**. Наберем код программы:

```
using System;
      using System.IO;                  // подключаем пространство имен System.IO
class Program
{ static void Main()
  { string s1 = "Привет";  int a = 15;  int b = 12;
                  // создаем объект класса StreamWriter, открываем файл на дозапись
    StreamWriter fw = new StreamWriter(@"D:\work\my.txt", true);
          // записываем строки 1 и 2
    fw.WriteLine("1: Работа с файлом");  fw.WriteLine("2: "+ s1);
          // добавляем строки 3, 4, 5
    fw.Write("3: a = " + a); fw.WriteLine(", b = " + b);      // строка 3
    fw.WriteLine("4: 'a + b' = " + a + b);                    // строка 4
    fw.WriteLine("5: a + b = " + (a + b));                    // строка 5
          fw.Close();                                  // закрываем записанный файл
                  // создаем объект класса StreamReader – открываем файл на чтение
    StreamReader fr = new StreamReader("D:\\work\\my.txt");
    string str; int i = 0;
    while ( (str = fr.ReadLine()) != null) Console.WriteLine("{0} -
         {1} ", ++i, str );
          fr.Close();                                  // закрываем прочитанный файл
  Console.ReadKey();
  }
}
```

2. Протестируем программу. Сравним варианты вывода.

Пример 2

*Составить программу, которая подсчитывает, выводит на консоль и записывает в файл **money.txt** (папка **Data**) ежемесячный баланс. Доходы вводят- ся еженедельно (4 раза за месяц), а количество источников расхода неизвестно (для завершения вводить три нуля).*

1. Создадим проект **con192**. Наберем код программы в теле метода **Main**:

```
// создаем объект класса  DirectoryInfo
DirectoryInfo di = new DirectoryInfo("Data");
// если папка уже существует, удаляем ее вместе с содержимым
if (di.Exists)  di.Delete(true);
// создаем новую папку Data в текущей с exe-файлом папке ( ..bin/debug/)
di.Create();
// создаем файл money.txt в папке Data (экранируем слеш)
StreamWriter sw= File.CreateText("Data\\money.txt");
int sumD = 0;                    // начальное значение суммы доходов за месяц
// вводим доходы за каждую из 4-х недель месяца
for (int i = 1; i < 5; i++)
  { Console.Write("Введите доход за {0} неделю: ", i); string
    debit = Console.ReadLine();
    sumD += int.Parse(debit);
  }
Console.WriteLine("=== Доход за месяц: " + sumD);      // вывод на консоль
     sw.WriteLine("=== Доход за месяц: " + sumD);      // запись в файл
Console.WriteLine();
      int sumR = 0; string credit = "";    // начальное значение суммы расходов
                        // организуем многократный ввод расходов (пока не введено три нуля 000)
while (credit != "000")
  { Console.Write("Введите расход "); credit =
    Console.ReadLine();
    sumR += int.Parse(credit);
  }
Console.WriteLine("=== Суммарный расход: " + sumR);    // вывод на консоль
        sw.WriteLine("=== Суммарный расход: " + sumR);    // запись в файл
Console.WriteLine();
int balans = sumD - sumR;                    // расчет и вывод баланса
string mes = (balans >= 0) ? "хорошо" : "плохо"; Console.WriteLine("===
Баланс: {0}. Это {1}!", balans.ToString(), mes);
     sw.WriteLine("=== Баланс: {0}. Это {1}!", balans.ToString(), mes);
sw.Close();                                  //закрываем файл money.txt
Console.ReadKey();
```

2. Протестируем программу. Проанализируем вывод даходов и расходов.

Рабочее задание

Выполнить инструкции приведенные ниже

Порядок выполнения работы

1. Модифицируйте программу **con192** так, чтобы в файл записывались все доходы и расходы.

2. Составьте программу, которая дописывает в файл **spis.txt** фамилию, имя и возраст студента.

3. Составьте программу, которая выводит на консоль и записывает в файл **week.txt** день недели, месяц и год.

4. Составьте программу, которая считывает из файла **stroki.txt** и выводит на консоль случайную пословицу.

5. Составьте программу, которая выводит на консоль и записывает в файл **tab.txt** таблицу умножения.

6. *Добавьте обработку исключений в программу **con192** (выводится сообщение «Повторите ввод», если введено не число).

7. Составьте программу, которая считывает из списка в файле **citata.txt** и выводит на консоль цитату дня по указанному номеру.

8. *Добавьте обработку исключений в программу задания 7 (выводит сообщение «Повторите», если введен номер больше, чем количество цитат в списке).

Контрольные вопросы

1. Как открыть и прочитать содержимое текстового файла в C#?
2. Как создать новый текстовый файл и записать в него данные в C#?
3. Как проверить существование файла перед его открытием или чтением в C#?
4. Как удалить файл с помощью C#?
5. Как переименовать файл с помощью C#?
6. Как скопировать файл с помощью C#?
7. Как переместить файл с помощью C#?
8. Как получить информацию о файле (например, имя, размер, дата создания и изменения) с помощью C#?
9. Как работать с директориями (создание, удаление, перемещение) с помощью C#?
10. Как записать данные в файл в формате CSV (разделенные запятыми значения) с помощью C#?

Практическая работа №15
Разработка приложений Windows Forms

Цель работы: формирование навыков создания приложений Windows Forms.

Теоретическая часть

Система Microsoft Visual Studio содержит удобные средства визуальной разработки Windows-приложений, которые позволяют в интерактивном режиме конструировать программы с графическим интерфейсом, используя готовые компоненты и шаблоны. В процессе разработки выделяют два основных этапа:

– **Визуальное проектирование** – создание внешнего облика приложения, которое заключается в помещении на форму компонентов (элементов управле- ния) и задании их свойств, а также свойств самой формы.

– **Программирование логики работы** приложения путем написания ме- тодов обработки событий.

Рассмотрим подробнее этапы разработки и структуру Windows-приложения.

– После запуска **MS Visual Studio** выбирается тип **Приложение Windows Forms** (Windows Forms Application) и шаблон **Visual C#**. Задается имя и распо- ложение проекта и решения, например, **myWin.**

– В результате открывается окно с формой в режиме конструктора (Design) (рис. 3.1). Слева по умолчанию располагается **Панель элементов** (Toolbox), а справа **Обозреватель решений** (Solution Explorer). При отсутствии их можно вызвать с помощью меню **Вид** (View).

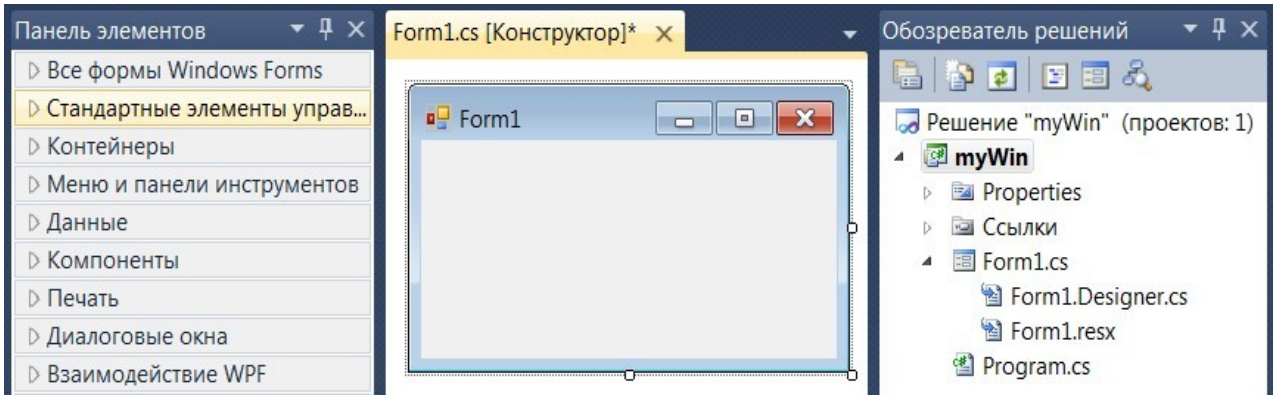


Рис. 3.1. Окно с формой в режиме конструктора

– Требуемые элементы (например, кнопка) перетаскиваются мышью с **Пане- ли элементов** на форму (рис. 3.2). Корректируется их расположение и размеры. На панели **Свойства** задаются их характеристики (имя, внешний вид, поведе- ние). Необходимые значения вводятся или выбираются из имеющихся в списке вариантов. Значок ▸ около имени свойства означает, что это свойство содер- жит другие, которые становятся доступными после щелчка на значке. При раз- мещении элемента на форме автоматически создается **экземпляр** соответству- ющего класса и шаблон программного кода.

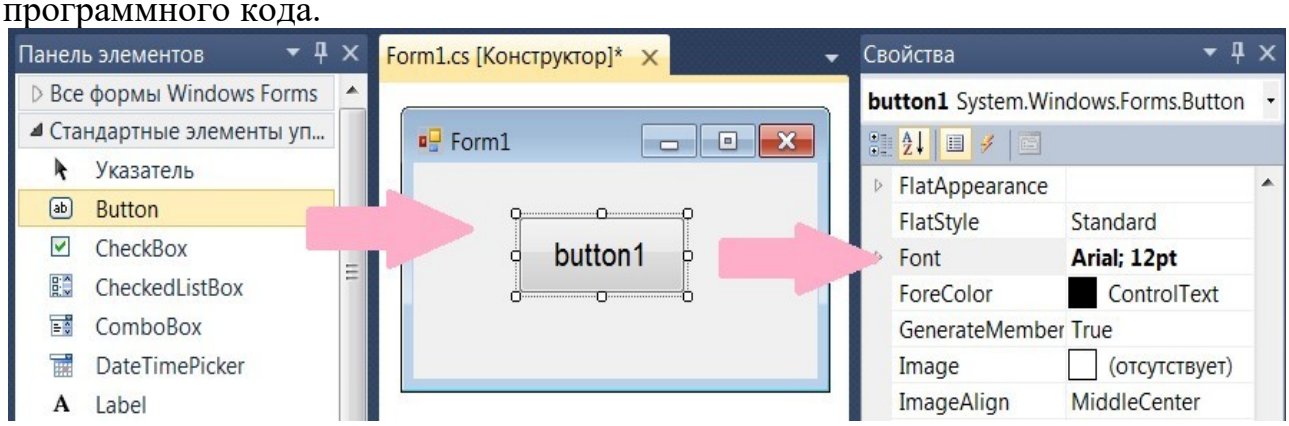


Рис. 3.2. Установка элемента управления и задание свойств

Проект Windows-приложения включает ряд файлов. Его структура отобра- жается на панели **Обозреватель решений**.

– Файл **Program.cs** содержит класс **Program**. Его метод **Main** является точкой входа, обеспечивает запуск приложения **Application.Run(new Form1())** и задает визуальный стиль (рис. 3.3). Для других целей при визуальном проек- тировании Windows-приложения его обычно **не** используют.

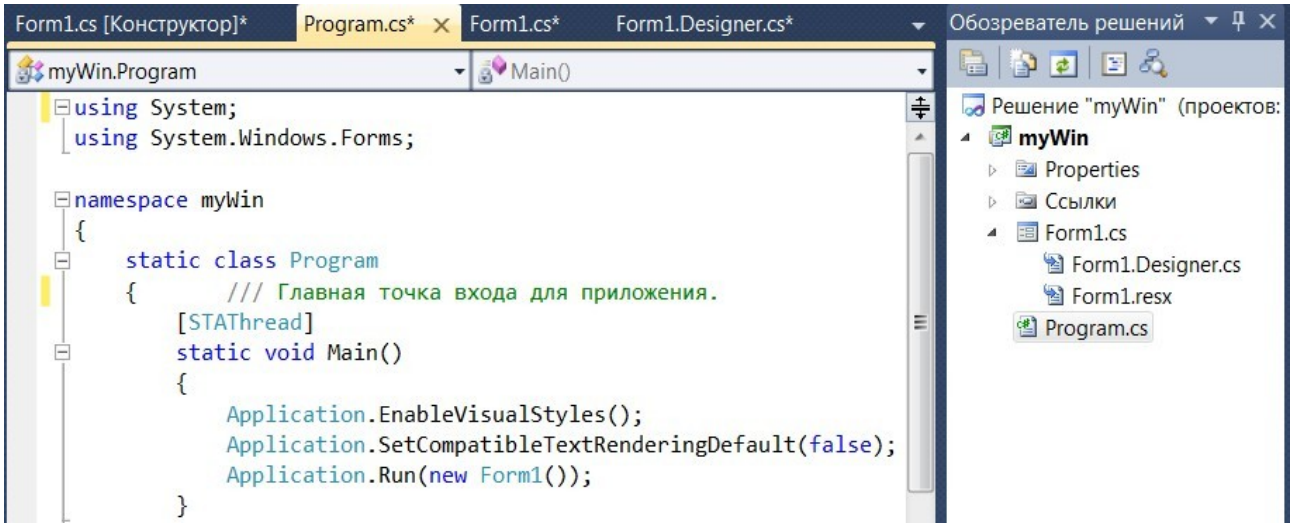


Рис. 3.3. Окна с кодом класса **Program**

Основной программный код с описанием используемых компонентов, объ- ектов, методов, а также обработчиков событий находится в классе **Form1**, ко- торый для удобства программирования разделен на две части (модификатор **partial** – частичный).

– Файл **Form1.cs** содержит часть класса **Form1** – конструктор с вызовом метода инициализации компонентов **InitializeComponent()** и обработчики со- бытий (рис. 3.4). Именно в обработчиках событий программируется логика ра- боты приложения.

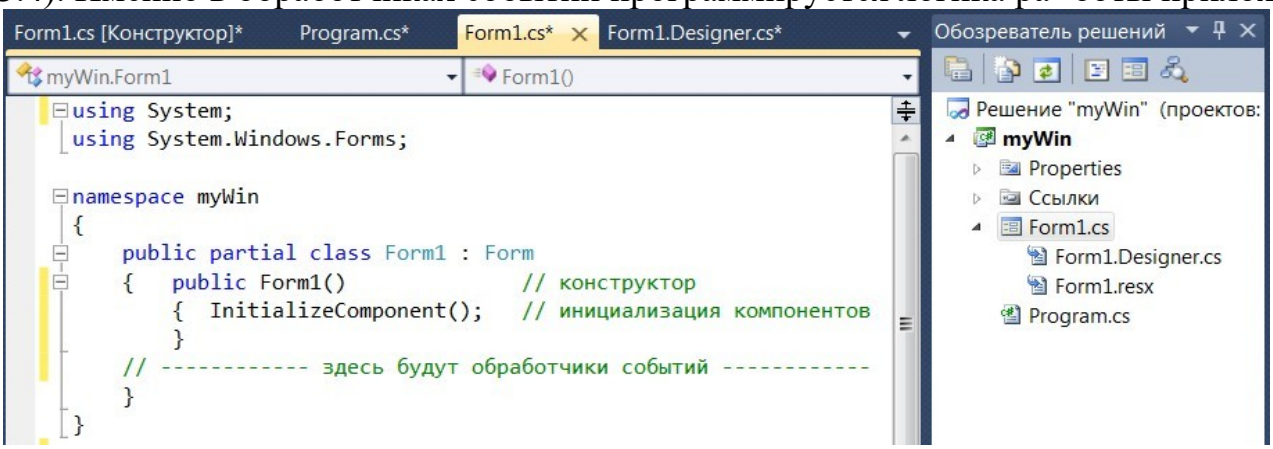


Рис. 3.4. Окно с программируемой частью кода класса **Form1**

– Файл **Form1.Designer.cs** в области #region … #endregion содержит код метода **InitializeComponent()**, автоматически создаваемый конструктором форм при установке элементов и регистрации событий (рис. 3.5).

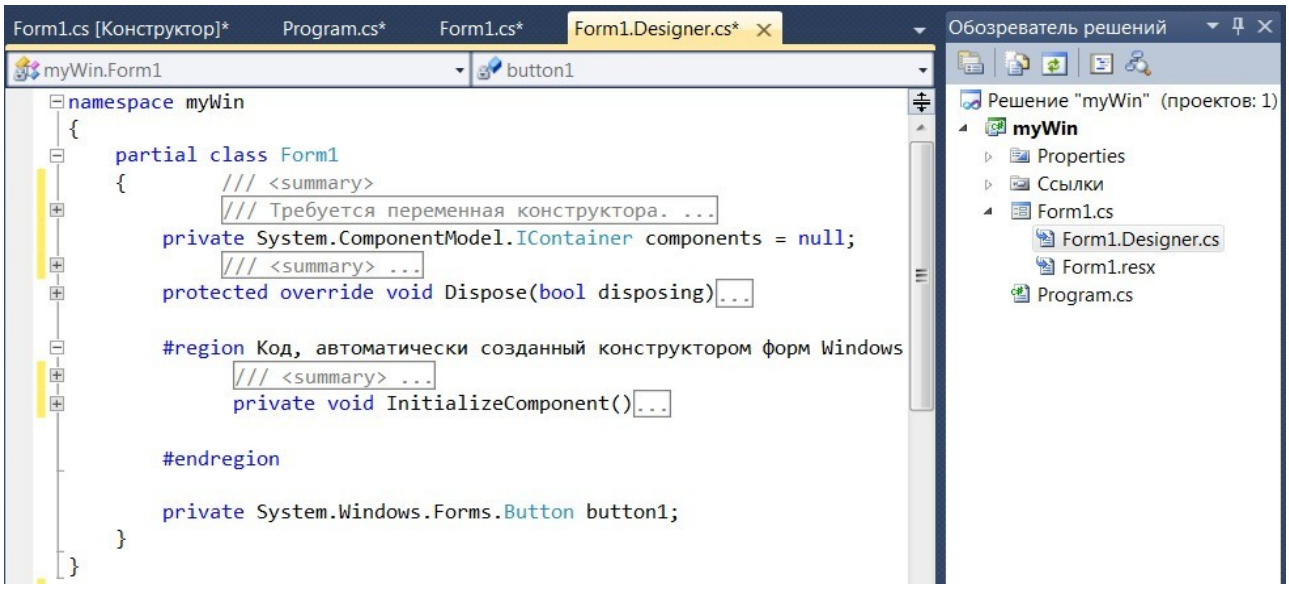


Рис. 3.5. Окно с автоматически создаваемым кодом класса **Form1**

Логика работы Windows-приложения основана на **объектно-событийной модели.** Определение поведения объектов начинается с принятия решений, ка- кие действия должны выполняться при нажатии кнопки, вводе текста, переме- щении курсора мыши, выборе пунктов меню, т. е. по каким событиям будут выполняться действия, реализующие функциональность программы. Для каж- дого класса определен свой набор событий, на которые он может реагировать. Нужное событие для выбранного объекта сначала необходимо зарегистриро- вать в методе **InitializeComponent()** (файл **Form1.Designer.cs**) или даже непо- средственно в конструкторе формы (файл **Form1.cs**), а затем запрограммиро- вать ответные действия в обработчике этого события.

Регистрацию события (подписку на событие) выполняют на вкладке **Со- бытия** (Events) панели **Свойств**а двойным щелчком мыши на поле, располо- женном справа от имени соответствующего события (рис. 3.6).

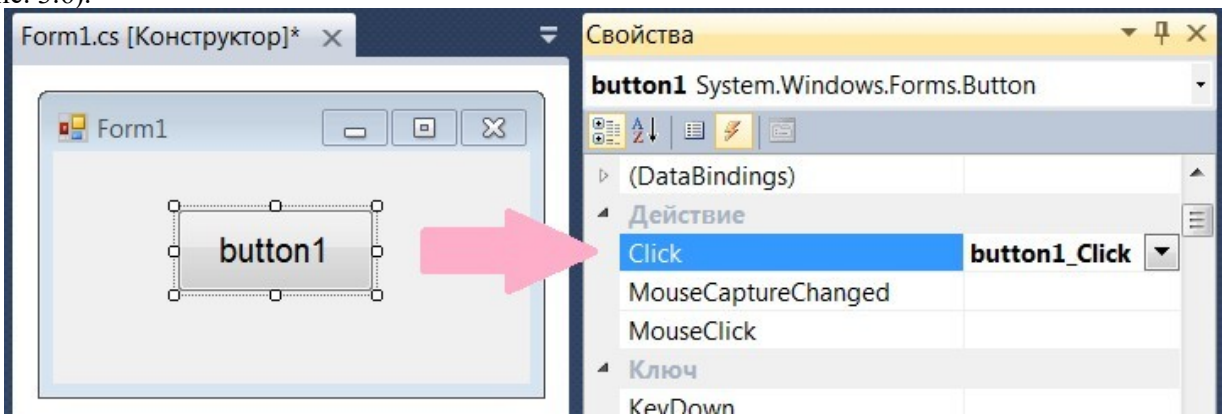


Рис. 3.6. Регистрация события нажатия кнопки Click

В методе **InitializeComponent()** (файл Form1.Designer.cs) появляется строка:

```
this.button1.Click += new System.EventHandler(this.button1_Click);
```

в файле Form1.cs автоматически создается шаблон соответствующего **метода** – **обработчика** (его имя формируется из имен объекта и события), в который предполагается вводить необходимый программный код. Обработчику переда- ются два параметра – объект-источник события и тип события.

```
private void button1_Click(object sender, EventArgs e)
{   ...  }
```

Итак, разработка Windows-приложений в системе MS Visual Studio сводит- ся к визуальному конструированию графического интерфейса в интерактивном режиме и программирование логики работы приложения путем написания ме- тодов обработки событий

Библиотека классов .NET включает пространство имен **System.Win- dows.Forms**, содержащее огромное количество типов строительных блоков Windows-приложений. Приведем лишь наиболее часто используемые с указа- нием некоторых свойств и событий:

– **Form** – форма, служит контейнером, на который устанавливаются все элементы;

– **Button** – кнопка; **Label** – надпись (метка) служит для вывода текста;

– **TextBox** – поле ввода-вывода текста; свойство **Text,** событие **TextChanged**;

– **CheckBox** – флажок, включатель; свойство **Checked**, событие **Checked-Changed**;

– **RadioButton** – переключатель, выбор одного варианта из нескольких; свойство **Checked**, событие **CheckedChanged**;

– **PictureBox** – контейнер для изображений (**bmp, jpg, gif, png**…); свойства: **Image** – объект типа **Image**, **Visible** – видимость (true – false), **SizeMode** – позици- онирование (Normal, StretchImage, Zoom, AutoSize).

Базовым классом (предком) для всех элементов управления является класс **Control**, который позволяет задавать общее поведение и свойства любого объ- екта графического интерфейса пользователя, например:

– **BackColor, ForeColor** – цвет фона и переднего плана;

– **BackgroundImage** – фоновый рисунок;

– **Text** – текст (строковые данные, ассоциированные с элементом); **Font** – шрифт;

– **Cursor** – вид курсора над элементом;

– **Enabled, Focused, Visible** – состояние элемента (true – false);
– **Opacity** – прозрачность элемента (0.0 прозрачный, 1.0 непрозрачный).

Основные **события** элементов управления:
– **Click, DoubleClick** – одинарный или двойной щелчки мышью;
– **MouseDown, MouseUp** – нажатие или отпускание кнопки мыши;
– **MouseMove** – перемещение мыши; **MouseHover** – мышь над элементом;
– **MouseEnter, MouseLeave** – мышь входит или покидает некоторую область;
– **KeyDown, KeyUp** – нажатие или отпускание любой клавиши;
– **KeyPress** – нажатие клавиши, имеющей ASCII-код;
– **DragDrop, DragEnter, DragLeave, DragOver** – события перетаскивания.

Положение, размеры и поведение элементов относительно контейнеров, в которые они вложены (например, **Form, Panel, GroupBox, PictureBox**), зада- ется свойствами **позиционирования**. Наиболее важные из них:
– **Location, Top, Left, Bottom, Right** – положение элемента;
– **Size, Width, Height** – размеры элемента;
– **Anchor** – привязка стороны элемента к сторонам контейнера. Если растя- гивать контейнер, то вместе с ним будет растягиваться и вложенный элемент (рис. 3.7, *а*). По умолчанию это свойство равно Top, Left.
– **Dock** – прикрепление элементf к определенной стороне контейнера (рис. 3.7, *б*). По умолчанию имеет значение None.

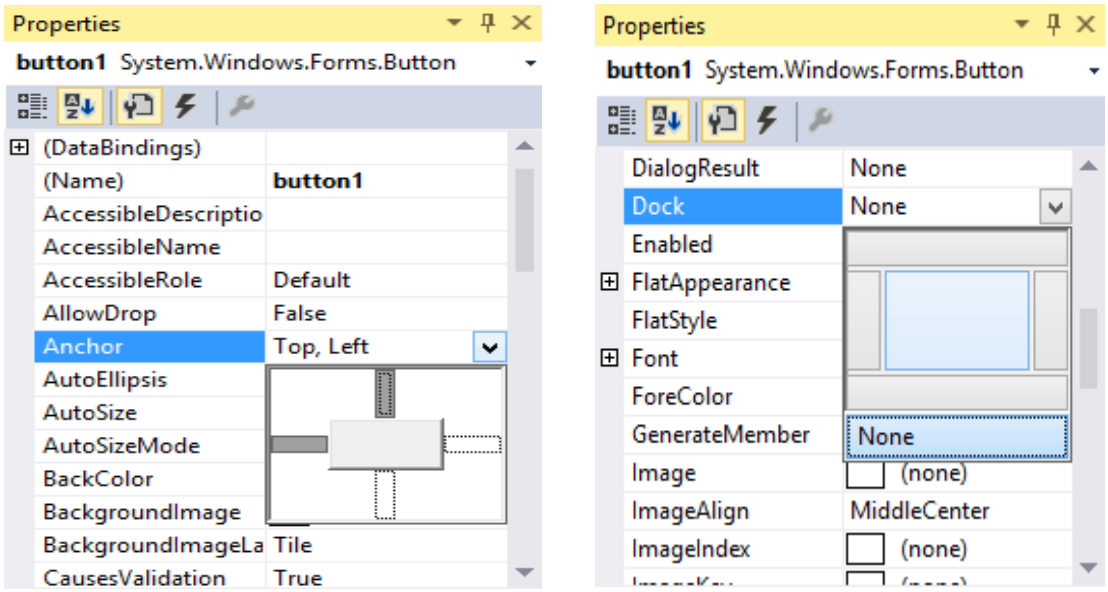


Рис. 3.7. Пример задания свойств Anchor (*а*) и Dock (*б*) для кнопки Button

Пример 1

Создать Windows-приложение, которое приглашает ввести в элемент TextBox имя и по нажатию кнопки выводит в элемент Label приветствие.

1. Запустим MS Visual Studio. Создадим новый проект.
2. Выберем тип приложения **Windows Forms** и шаблон **Visual C#**.
3. В поле ввода **Расположение (Location)** зададим рабочую папку, в кото- рой будет сохраняться проект. Введем имя проекта, например: **wf311**.
4. Откроется окно с формой **Form1** в режиме конструктора. По умолчанию слева располагается **Панель элементов** (Toolbox), а справа **Обозреватель ре- шений** (Solution Explorer).
5. Мышью перетащим с **Панели элементов** на форму две надписи **Label**, поле ввода текста **TextBox** и кнопку **Button**. При этом будут созданы **экземп- ляры** объектов, которым по умолчанию присваиваются имена соответствую- щих классов (с малой буквы) с номерами: **label1, label2, textBox1, button1**.

6. По очереди выделяем установленные элементы и на панели **Свойства** изменяем предлагаемые по умолчанию значения свойства **text:** у формы на «**Приветствие**», у надписи_1 на «**Введите имя**», у кнопки на «**Нажмите**» **(**рис. 3.8). Подберем размеры шрифта (свойства **Font** 10–12 пт.).

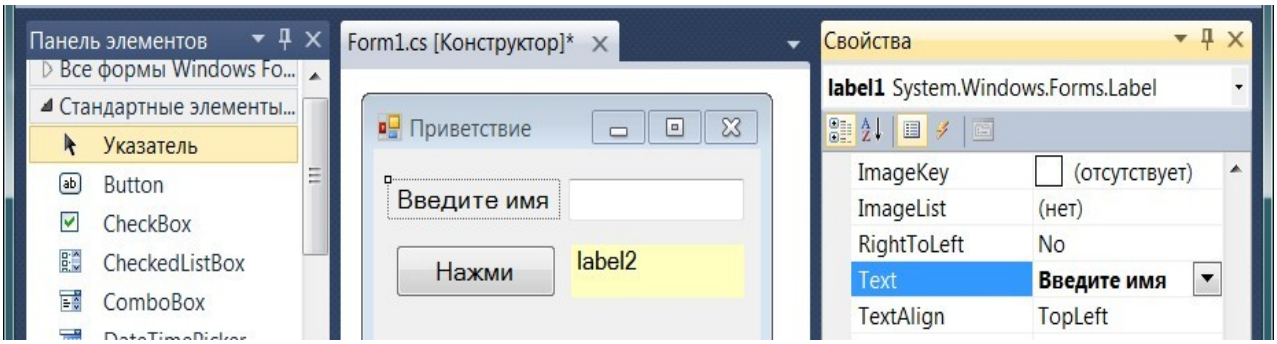


Рис. 3.8. Задание свойств элементов формы

7. Зарегистрируем событие нажатия кнопки. Для этого выделим кнопку и на вкладке **События** (**Events**) панели свойств выберем **Click**. Заметим, что регистрировать события, связанные с элементами по умолчанию, можно и двой- ным щелчком мыши по выбранному элементу.

8. Откроется окно **Form1.cs**, в котором автоматически будет создан шаб- лон обработчика. Введем в него код:

```
private void button1_Click(object sender, EventArgs e)
{  label2.Text = "Привет, " + textBox1.Text;
}
```

9. Протестируем программу. Результат может выглядеть так (рис. 3.9, *а*). Откорректируем программный код и свойства элементов (например, зададим желтый фон **BackColor** надписи_2).

10. Модифицируем наш проект. Разместим на форме элемент **pictureBox1** (рис. 3.9, *б*)**.** Импортируем изображение (свойство **Image**) из файла **hacker.jpg**. Подберем размеры и установим значения свойств **Visible = false** (в исходном состоянии изображение невидимо) и **SizeMode = StretchImage** (растягивается по размеру контейнера).

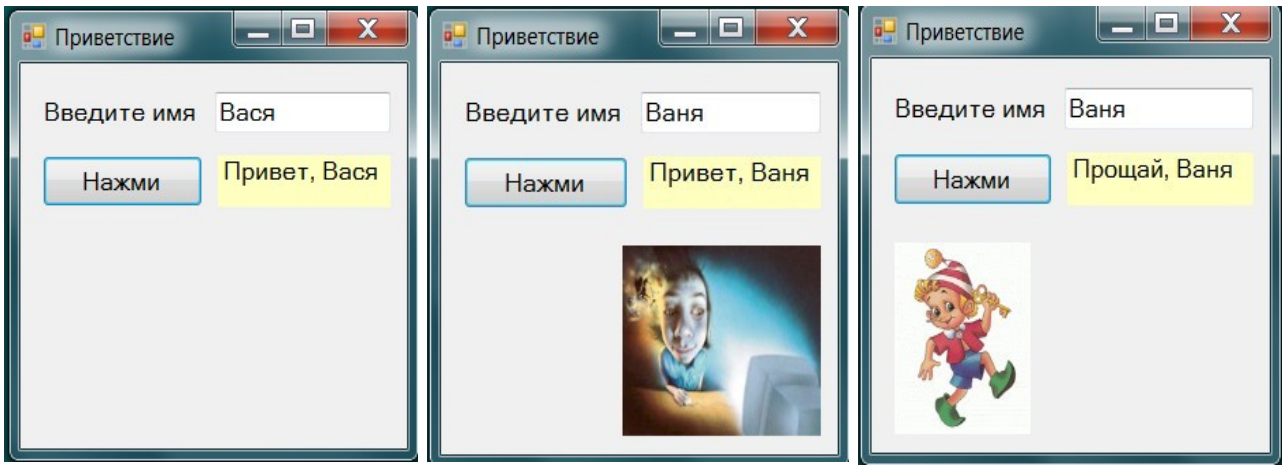


Рис. 3.9. Модификация интерфейса приложения

11. В обработчик события кнопки добавим код:

```
pictureBox1.Visible = true;
```

12. Протестируем программу. Теперь при вводе имени и нажатии кнопки появляется приветствие и изображение.

13. Еще раз модифицируем проект так, чтобы по щелчку мыши на **picture- Box1**

это изображение исчезало, а появлялась новое из файла **buratino.gif**, и надпись «**Привет ,**» заменялась на «**Прощай ,**» (рис. 3.9, *в*).

14. Разместим на форме элемент **pictureBox2** и импортируем в него изображение **buratino.gif**. Настроим размеры и свойства.

15. Зарегистрируем событие **MouseClick** (щелчок мыши) на **pictureBox1**.

В созданный шаблон обработчика введем код:

```
private void pictureBox1_MouseClick(object sender, MouseEventArgs e)
 { label2.Text = "Прощай, " + textBox1.Text;
   pictureBox1.Visible = false;
   pictureBox2.Visible = true;
 }
```

16. Протестируем программу. Откорректируем код и свойства элементов.

Пример 2

Простейший тест с использованием элементов ***radioButton.*** *Выбор единственного верного ответа.*

1. Создадим новый проект **wf312** типа Windows Forms.

2. Разместим на форме надпись **label1** с вопросом: «Что такое компьютер- ный вирус?», три элемента **radioButton** с ответами «Вредный программист», «Вредоносная программа», «Живущий в компьютере микроб», а также кнопку **button1** «ответ» и поле **textBox1** для вывода результата (рис. 3.10).

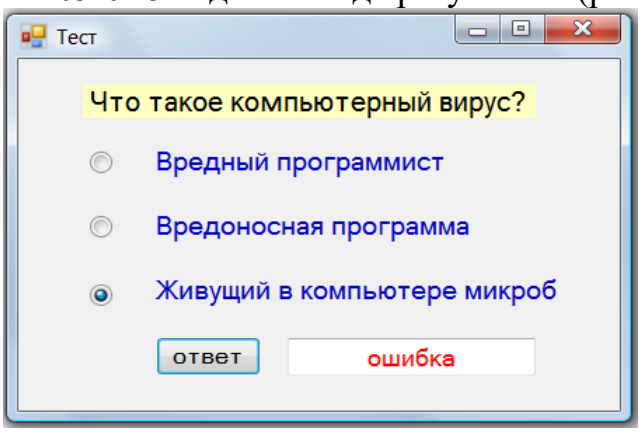


Рис. 3.10. Интерфейс теста с элементами **radioButton**

3. Зарегистрируем событие нажатия кнопки. В шаблон обработчика вве- дем код:

```
private void button1_Click(object sender, EventArgs e)
 {  textBox1.Text = (radioButton2.Checked) ? "верно" : "ошибка";
 }
```

Пример 3

Тест с использованием элементов ***checkBox.*** *Выбор нескольких верных ответов.*

1. Создадим новый проект **wf313** типа Windows Forms.

2. Разместим на форме надпись **label1** «Устройства ввода:», четыре эле- мента **checkBox** с текстами ответов «клавиатура», «рука», «мышь», «кошка», кнопку «ответ» и надпись **label12** для вывода результата (рис. 3.11).

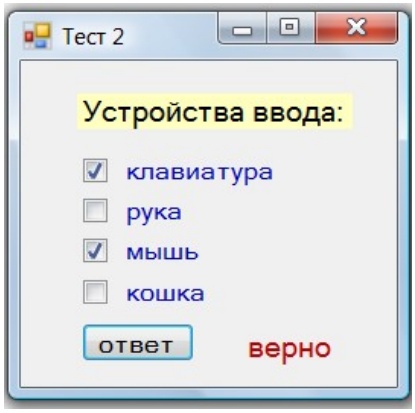


Рис. 3.11. Интерфейс теста с элементами **checkBox**

3. Зарегистрируем событие нажатия кнопки. В шаблон обработчика вве- дем код:

```
private void button1_Click(object sender, EventArgs e)
 {  label2.Text = (checkBox1.Checked && checkBox3.Checked &&
    !checkBox2.Checked && !checkBox4.Checked) ? "верно" : "ошибка";
 }
```

Порядок выполнения работы

Создайте Windows-приложения, в которых выполняются следующие действия.

Рабочее задание

1. При щелчке мыши по форме показывается изображение автомобиля avto.gif. Щелчок мышью по изображению скрывает его.

2. При наведении указателя мыши на форму цвет ее фона становится жел- тым. При уводе указателя мыши с формы ее цвет становится серым.

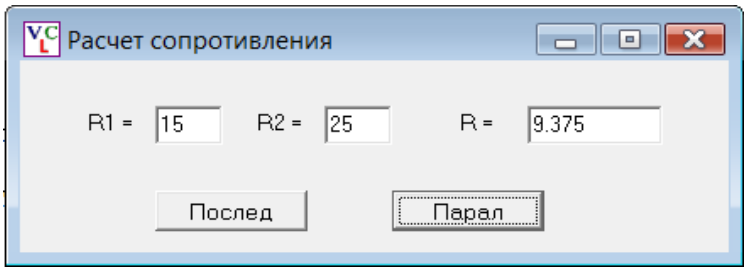
3. При нажатии и удержании левой кнопки мыши на форме ее цвет стано- вится розовым. Отпускание кнопки мыши изменяет цвет фона на голубой.

4. По нажатию кнопки проверяется правильность введенного в текстовое поле пароля. При совпадении с заданным ключом выводится «Добро пожало- вать» и появляется изображение Буратино (файл buratino.gif), при несовпадении выводится «Вход запрещен» и изображение попугая (файл popugai.gif).

5. Вычисляется среднее арифметическое **sred** двух введенных чисел **a** и **b**. Ввод в текстовые поля, вывод по нажатию кнопки в надпись.

6. Вычисляется площадь **s** и периметр **p** прямоугольника по сторонам **a** и **b**. Ввод в текстовые поля, вывод по нажатию кнопки в надписи.

7. Вычисляется сопротивление при последовательном или параллельном соединении резисторов. Ввод и вывод в текстовые поля по нажатию кнопок.



8. *Создайте калькулятор, выполняющий 5 действий. Ввод и вывод в текс- товые поля. Счет по нажатию кнопок.

Контрольные вопросы

1. Что такое Windows Forms и какую роль они играют в разработке приложений на платформе Windows?
2. Как создать новый проект Windows Forms в Visual Studio с использованием C#?
3. Как добавить элемент управления (например, кнопку или текстовое поле) на форму в приложении Windows Forms?
4. Как связать событие клика на кнопку с выполнением определенного действия (метода) в коде C#?
5. Как изменить свойства элемента управления (например, размер, цвет фона) в коде C# приложения Windows Forms?
6. Как добавить и настроить меню в приложении Windows Forms?
7. Как добавить и использовать диалоговые окна (например, окно открытия файла или сообщение об ошибке) в приложении Windows Forms?
8. Как обрабатывать ввод пользователя (например, нажатия клавиш или ввод текста) в приложении Windows Forms?
9. Как работать с многооконными приложениями в Windows Forms (например, создать новое окно или переключаться между окнами)?
10. Как развернуть и распространить приложение Windows Forms для использования на других компьютерах?

Практическая работа №16
Интерактивное управление параметрами приложений

Цель работы: формирование навыков создания Windows-приложений с интерактивным управлением параметрами.

Теоретическая часть

Библиотека классов .NET содержит элементы, которые можно использо- вать для управления параметрами Windows-приложений в интерактивном ре- жиме без текстового ввода. Приведем примеры таких элементов и событий, позволяющих отслеживать изменения их свойств.

Элемент **TrackBar** позволяет с помощью ползунка изменять числовые зна- чения свойства **Value** типа **int** из диапазона Minimum, Maximum. Событие **Scroll.**

NumerickUpDown позволяет изменять значение **Value** числового типа **decimal** (десятичная дробь с разделителем, заданным локализацией операцион- ной системы, например, запятая в русскоязычных ОС). Событие **ValueChanged.** Приращение **Increment** может как целым, так и дробным. Отображаемое коли- чество десятичных знаков задается **DecimalPlaces** (по умолчанию **0**).

Следующие три элемента предназначены для выбора из списков.

– **DomainUpDown** возвращает строку (свойства **Text** типа **string)**. Событие **TextChanged**.

– **ListBox** (список) и **ComboBox** (поле с выпадающим списком) возвраща- ют индекс (свойство **SelectedIndex**). Событие **SelectedIndexChanged**.

Пример 1

*Вычисление площади круга. Использование элемента **trackBar**.*

1. Создадим проект **wf321** типа Windows Forms.

2. Разместим на форме две надписи с текстами **“R = ”** и **“S = ”**, два тексто- вых поля и элемент **trackBar** (рис. 3.12)**.** Установим его свойства Minimum = 10, Maximum = 80.

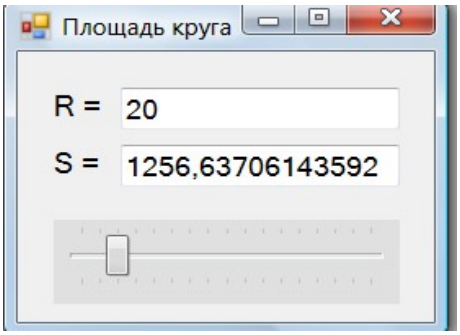


Рис. 3.12. Интерфейс приложения с элементами **label, textBox** и **trackBar**

3. Выделим элемент **trackBar.** Зарегистрируем событие перемещения пол- зунка **Scroll.** В шаблон обработчика введем код:

```
private void trackBar1_Scroll(object sender, EventArgs e)
  {  int r = trackBar1.Value; textBox1.Text =
     Convert.ToString(r);
     textBox2.Text = Convert.ToString(Math.PI*r*r);
  }
```

4. Протестируем программу. Откорректируем код и свойства элементов.

Пример 2

Расчет силы тока. Использование элемента ***numericUpDown****.*

Напряжение и сопротивление будем задавать элементами ***numericUpDown****, силу тока вычислять по формуле* ***I = U/R*** *и выводить в текстовое поле по на- жатию кнопки.*

1. Создадим проект **wf322** типа Windows Forms.

2. Разместим на форме две надписи с текстами **“U = ”** и **“R = ”,** кнопку **“I = ”**, текстовое поле и два элемента **numericUpDown** (рис. 3.13). Установим их свой- ства Minimum = 1, Maximum = 20, Increment = 1.

3.

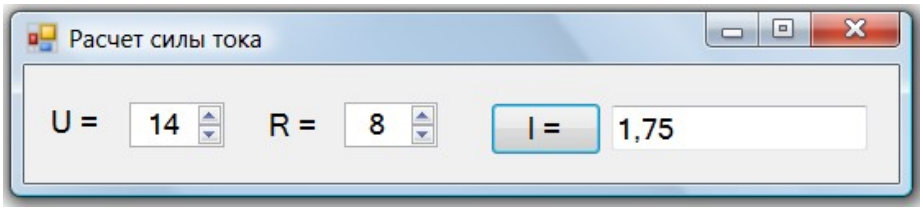


Рис. 3.13. Интерфейс приложения с кнопкой и элементами **numericUpDown**

4. Зарегистрируем событие нажатия кнопки. В шаблон обработчика вве- дем код:

```
private void button1_Click(object sender, EventArgs e)
 {  decimal U = numericUpDown1.Value;  decimal R = numericUpDown2.Value;
    textBox1.Text = Convert.ToString(U/R);
 }
```

5. Протестируем программу, изменяя напряжение U и сопротивление R.

Заметим, что хотя параметры U и R задаются элементом **numericUpDown,** рас- чет и вывод результата выполняется по нажатию кнопки.

Во многих практических задачах вычисления и вывод результатов полезно выполнять непосредственно при изменении входных параметров. Для этого ис- пользуют события, отслеживающие изменения свойств элементов. Так, в при- мере 1 использовано событие **Scroll** элемента **trackBar.** Рассмотрим использо- вание двух событий **ValueChanged** элементов **numericUpDown.**

Пример 3

Расчет сопротивления при параллельном соединении резисторов. Использование событий двух элементов ***numericUpDown.***

При параллельном соединении резисторов складываются величины, обрат- ные их сопротивлениям ***1/R = 1/R1 + 1/R2****.*

1. Создадим проект **wf323** типа Windows Forms.

2. Разместим на форме три надписи **“R1 = ”, “R2 = ”, “R = ”,** текстовое по- ле и два элемента **numericUpDown** (рис. 3.14). Установим свойства: Minimum = 1, Maximum = 80, Increment = 0,1, DecimalPlaces = 1.

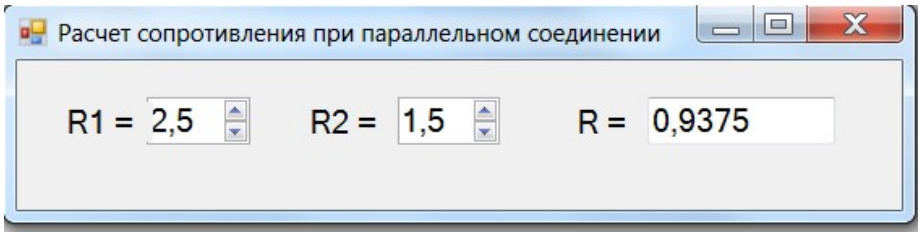


Рис. 3.14. Интерфейс приложения с элементами **numericUpDown**

3. Зарегистрируем события **ValueChanged** двух элементов **numericUpDown**.

4. В шаблоны обработчиков введем коды. Для их упрощения вне обработ- чиков объявлены и инициализированы поля r1 и r2, а вычисление и вывод вы- несены в метод **ShowR()**, который вызывается в обработчиках.

```
public decimal r1 = 1; public decimal r2 = 1; public
void ShowR()
  {  textBox1.Text = Convert.ToString(1/(1/r1 + 1/r2));  }
private void numericUpDown1_ValueChanged(object sender, EventArgs e)
  {  r1 = numericUpDown1.Value;  ShowR();  }
private void numericUpDown2_ValueChanged(object sender, EventArgs e)
  {  r2 = numericUpDown2.Value;  ShowR();  }
```

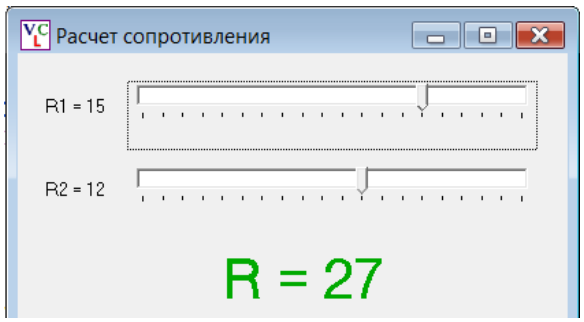
5. Протестируем программу. Теперь при любом изменении сопротивлений резисторов результат сразу же пересчитывается.

Порядок выполнения работы

Создайте приложения Windows Forms, которые вычисляют и выводят:

Рабочее задание

1. Сопротивление при последовательном (параллельном) соединении рези- сторов. Вывод в надписи по событиям **Scroll** элементов **trackBar**.



2. Площадь поверхности **s** и объем шара **v** по радиусу **r**. Ввод и вывод в текстовые поля по события **Scroll** элемента **trackBar**.

3. Высоту $\mathbf{h = V^2/2g}$ подъема мяча, брошенного вертикально вверх с на- чальной скоростью **V** (задается элементом **trackBar**). Вывод в надпись по на- жатию кнопки.

4. Потенциальную энергию **E = mgh** камня массой **m** (ввод в текстовое по- ле) на высоте **h** (задается элементом **trackBar**). Вывод в надпись по нажатию кнопки.

5. Путь **s = v*t**, пройденный автомобилем за время **t**. Скорость **v** вводится в текстовое поле, время **t** (10 – 40) задается элементом **trackBar.** Вывод в над- пись по событию **Scroll** элемента **trackBar**.

6. Оплату за электроэнергию = тариф * расход. Вывод в надпись по собы- тию **Scroll.** Тариф (в руб за 1 кВт ч) задается элементом **numericUpDown.** Рас- ход в кВт ч (от 0 до 400) задается элементом **trackBar**,

7. Стоимость поездки на автомобиле (*ввод*: **s** – расстояние, **b** – расход бен- зина на 100 км, **c** – цена бензина за 1 литр). Вывод в надпись по события **Scroll** элемента **trackBar.**

8. Стоимость товара в трех валютах по его стоимости в бел. рублях. Ввод в **textBox** (бел. руб.), вывод по нажатию кнопки в надписи. Курсы валют зада- ются элементами **numericUpDown.**

Контрольные вопросы

1. Как создать окно приложения с интерактивными элементами управления в C#?
2. Как добавить кнопку в окно приложения и связать ее с определенным методом в C#?
3. Как создать текстовое поле, чтобы пользователь мог вводить значения параметров в окне приложения в C#?
4. Как связать элементы управления с переменными, чтобы изменения пользовательских параметров отражались в коде C#?
5. Как добавить ползунок для выбора числового значения параметров в окне приложения в C#?
6. Как добавить выпадающий список для выбора определенных значений параметров в окне приложения в C#?
7. Как создать чекбоксы для включения/отключения определенных параметров в окне приложения в C#?
8. Как создать группу радиокнопок для выбора одного значения из нескольких в окне приложения в C#?
9. Как обрабатывать события взаимодействия пользователя с элементами управления в окне приложения в C#?
10. Как использовать значения параметров, выбранные пользователем, в коде приложения в C#?

Практическая работа №17
Использование таймера. Анимация

Цель работы: формирование навыков использования таймера и создания простейшей анимации движения.

Теоретическая часть

Неотображаемый на форме компонент **Timer** предназначен для запуска периодически повторяющихся действий. Свойство **Interval** задает период (в мил- лисекундах), с которым будет повторяться событие **Tick**. При установке свой- ства **Enabled = true** таймер включается вместе с запуском приложения. Метод **Start()** запускает, а **Stop()** останавливает таймер.

Типичные примеры использования таймера – часы и секундомер (вывод времени и даты), а также анимация (имитация плавного изменения положения, размеров и формы объектов).

Пример 1

Простые часы. Вывод времени и даты по таймеру.

1. Создадим новый проект **wf331** типа Windows Forms.

2. Разместим на форме размером 540 × 230 две надписи и зададим их свойства:

– **label1** для вывода времени (свойства: **BackColor = Green**, **ForeColor = = Yellow**, **Text = 00:00:00**, размер шрифта **Font = 72**);

– **label2** для вывода даты (**ForeColor = Olive**, **Text = дата**, **Font = 16**).

3. Из категории **Компоненты** (Components) панели элементов перетащим на форму **Timer**. Его значок отобразится в нижней части окна **Конструктор** (рис. 3.15)**.** Зададим его свойства: **Enabled = true, Interval = 100.**

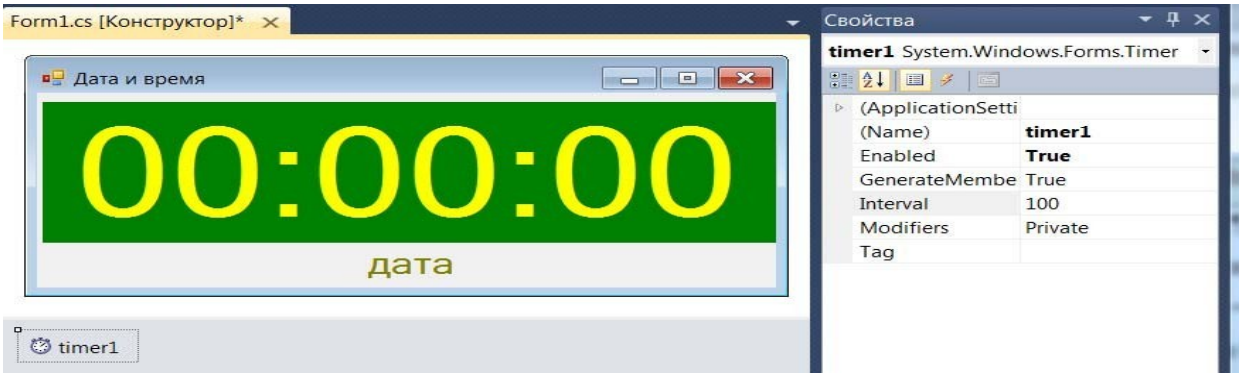


Рис. 3.15. Настройка элементов формы и таймера

4. Зарегистрируем событие **Tick** таймера. В шаблон обработчика введем код:

```
private void timer1_Tick(object sender, EventArgs e)
    {  label1.Text = DateTime.Now.ToLongTimeString();        // вывод времени
       label2.Text = DateTime.Now.ToLongDateString();        // вывод даты
 }
```

5. Протестируем программу (рис. 3.16, *а*). При необходимости откорректируем свойства компонентов и программный код.

Рис. 3.16. Простые часы (*а*) и секундомер (*б*)

Пример 2

Секундомер. Вывод минут и секунд.

1. Создадим новый проект **wf332** типа Windows Forms**.**

2. Разместим на форме размером 380×200 надпись **label1** (свойства **BackColor = Green, ForeColor = Yellow, Text = 0:0.0, Font = 60**) и две кнопки **Старт** и **Сброс** (рис. 3.16, *б*).

3. Перетащим на форму **Timer** (свойства **Enabled = false, Interval = 100**).

4. Зарегистрируем события **Tick** таймера, а также нажатий кнопок **Click**.

В шаблоны обработчиков введем коды:

```
private int m, s, ms;                                    // объявление полей
      // нажатие кнопки Старт/Стоп
private void button1_Click(object sender, EventArgs e)
 { if (timer1.Enabled)
    {  timer1.Stop();  button1.Text = "Старт";  }    // остановка таймера
```

```
   else {  timer1.Start();  button1.Text = "Стоп";  } // запуск таймера
 }
      // нажатие кнопки Сброс, обнуление значений
private void button2_Click(object sender, EventArgs e)
 {  m = 0; s = 0; ms = 0;  label1.Text = "0:0.0"; }
      // счет и вывод по событию Tick таймера
private void timer1_Tick(object sender, EventArgs e)
 {  ms++;  s = ms/10;  m = s/60;
    label1.Text = m + ":" + s%60 + "." + ms%10;
 }
```

5. Протестируем программу. Откорректируем код и свойства компонентов.

Пример 3

Простейшая анимация движения.

Создадим новый проект **wf333** типа Windows Forms.

Разместим на форме размером 580 × 200 две кнопки **Старт** и **Стоп**, а также элемент **PictureBox** размером 140 × 100. Импортируем в него изображение из файла **beg.gif** (рис. 3.17, *а*).

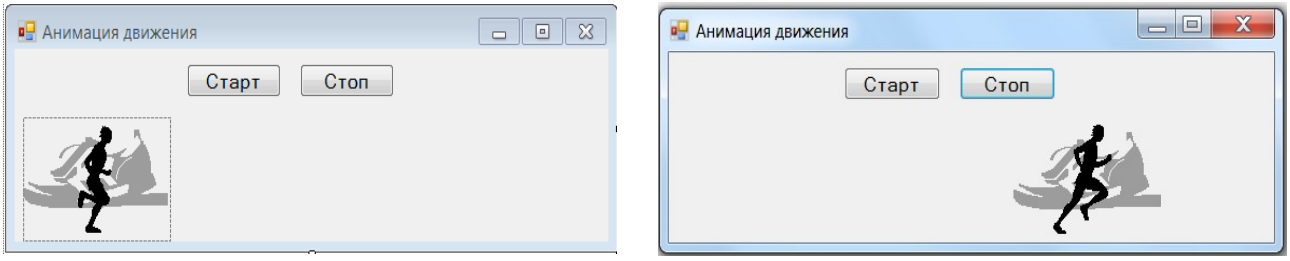


Рис. 3.17. Начальная (*а*) и промежуточная фазы анимации (*б*)

3. Перетащим на форму **Timer** (свойства **Enabled = false, Interval = 20)**.

4. Зарегистрируем события **Tick** таймера, а также нажатий кнопок **Click**.

В шаблоны обработчиков введем коды:

```
// перемещение по тикам таймера вправо на 4px, если Left <520, иначе в начало
private void timer1_Tick(object sender, EventArgs e)
 { if (pictureBox1.Left < 520) pictureBox1.Left += 4; else
    pictureBox1.Left = 8; }
private void button1_Click(object sender, EventArgs e)
 {  timer1.Enabled = true;  }                          // старт
private void button2_Click(object sender, EventArgs e)
 {  timer1.Enabled = false;  }                         // стоп
```

5. Протестируем программу.

Рабочее задание

Выполнить инструкции приведенные ниже

Порядок выполнения работы

1. Создайте секундомер с одной кнопкой **Сброс**, который запускается и останавливается щелчком мыши по форме.

2. Создайте приложение, которое по введенной дате (год, месяц и число) показывает день недели.

3. Создайте часы, которые показывают, сколько дней (часов, минут) оста- лось наступления нового года.

4. Создайте приложение, в котором анимируется прямолинейное движение

спутника **sputnik.jpg** на фоне звездного неба **sky.gif**.

5. Создайте приложение, в котором анимируется падение яблока **apple.gif** с башни **tower.jpg**.

6. Создайте приложение, в котором по щелчку мышью по изображению совы **sova.jpg** оно начинает увеличиваться до достижения двукратного размера. Щелчок мыши по увеличенному изображению вызывает его уменьшение до первоначальных размеров.

7. Создайте приложение, в котором анимируется движение мяча **football.gif**, брошенного под углом к горизонту на фоне деревьев **trees.gif**. Траектория дви- жения парабола.

8. Создайте приложение, в котором анимируется движение Луны **luna.gif** вокруг Земли **zem.gif** по эллиптической траектории.

Контрольные вопросы

1. Как создать и использовать таймер в C# для выполнения определенных действий с определенной задержкой?
2. Какой класс нужно использовать для работы с таймером в C#?
3. Как настроить интервал работы таймера в C#?
4. Как связать событие срабатывания таймера с определенным методом в C#?
5. Как создать простейшую анимацию движения с использованием таймера в C#?
6. Как изменять координаты объекта для создания анимации движения в C#?
7. Как использовать таймер в цикле для создания плавной анимации движения в C#?
8. Как задать скорость и направление движения объекта в анимации с использованием таймера в C#?
9. Как остановить и возобновить работу таймера в C#?
10. Как обрабатывать события начала и окончания анимации в C#?

Практическая работа №18
Использование меню и диалоговых окон

Цель работы: формирование навыков создания меню и диалоговых окон.

Теоретическая часть

Библиотека .NET содержит компоненты позволяющие проектировать при- ложения с развитым графическим интерфейсом, включающим меню, панели инструментов, разнообразные диалоговые окна. Некоторые из них сразу отоб- ражаются на форме, другие могут вызываться.

Основные компоненты для построения меню:

ToolStrip – панель инструментов (базовый класс – контейнер). На ней размещают элементы – объекты **ToolStripItem**.

Наследник **ToolStrip** – полоска меню **MenuStrip**.

StatusStrip – строка (полоска) состояния.

Основные свойства элементов меню: вид, размеры, позиционирование (Dock, LayoutStyle), видимость (ShowItemToolTips). Основные события связаны с вы- бором конкретного пункта меню щелчком мыши и имеют синтаксис:

пунктToolStripMenuItem_Click.

Важнейшей особенностью Windows-приложений является **оконный ин- терфейс**, включающий окна разного типа, имеющие свое назначение, функцио- нал и внешний вид. Каждое приложение имеет одно **главное окно**. Напомним, что класс главного окна приложения содержит точку входа в приложение (ста- тический метод **Main**). При закрытии главного окна приложение завершается. **Модальное окно** не позволяет пользователю переключаться на другие окна того же приложения, пока не будет завершена работа с текущим окном. **Немодаль- ное окно** позволяет переключаться на другие окна. В виде модальных обычно оформляют **диалоговые окна**, требующие от пользователя ввода какой-либо информации или подтверждения.

Наиболее часто используют **Окно сообщений** (класс **MessageBox**), кото- рое вызывается методом **Show()** и может принимать ряд параметров (рис 3.18): **text** – текст выводимого сообщения, **caption** – текст заголовка окна сообщения, **icon** – значок окна сообщения, **buttons** – используемые кнопки.

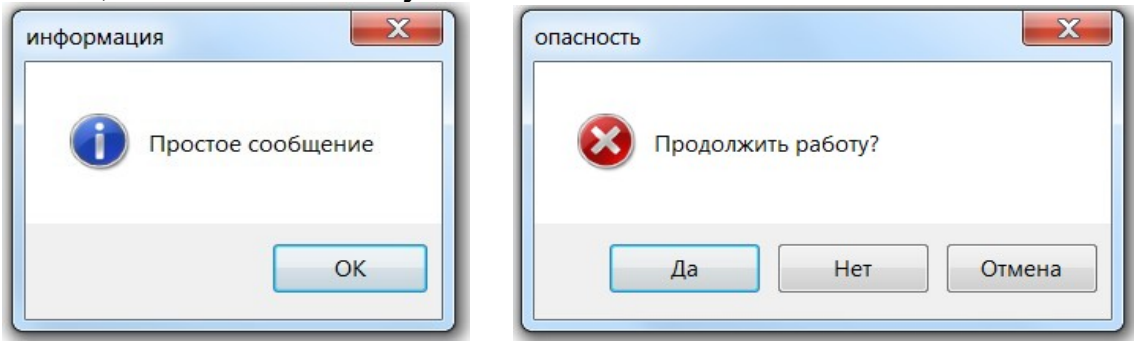


Рис. 3.18. Вид диалогового окна сообщений **MessageBox**

Это окно может иметь от 1 до 3 кнопок (рис. 3.18). Количество и назначе- ние задается значением свойства MessageBoxButtons: **OK** – одна кнопка OK; **OKCancel** – две кнопки (OK и Отмена); **YesNo** – две кнопки (Да и Нет); **YesNoCancel** – три кнопки (Да, Нет, Отмена). Метод **Show()** возвращает объект DialogResult, позволяющий узнать, какая кнопка в окне была нажата. Вид знач- ков в соответствии с содержанием сообщения можно задавать значением свой- ства MessageBoxIcon: **Information** – буква i в кружке; **Error** – белый знак «X» в красном круге; **Warning** – восклицательный знак в желтом треугольнике; **Question** – вопросительный знак в круге.

Выполнение многошаговых операций значительно упрощается благодаря использованию **специализированных** диалоговых окон (или просто диалогов), имеющих набор требуемых свойств, методов и событий. Эти окна отображают- ся методом **ShowDialog().** Приведем примеры таких диалогов.

– **ColorDialog** – выбор цвета (рис. 3.19, *а*). Возвращает выбранный цвет:
colorDialog1.ShowDialog(); this.BackColor = colorDialog1.Color;

– **FontDialog** – выбор шрифта (рис. 3.19, *б*). Возвращает выбранный шрифт:
fontDialog1.ShowDialog(); textBox1.Font = fontDialog1.Font;

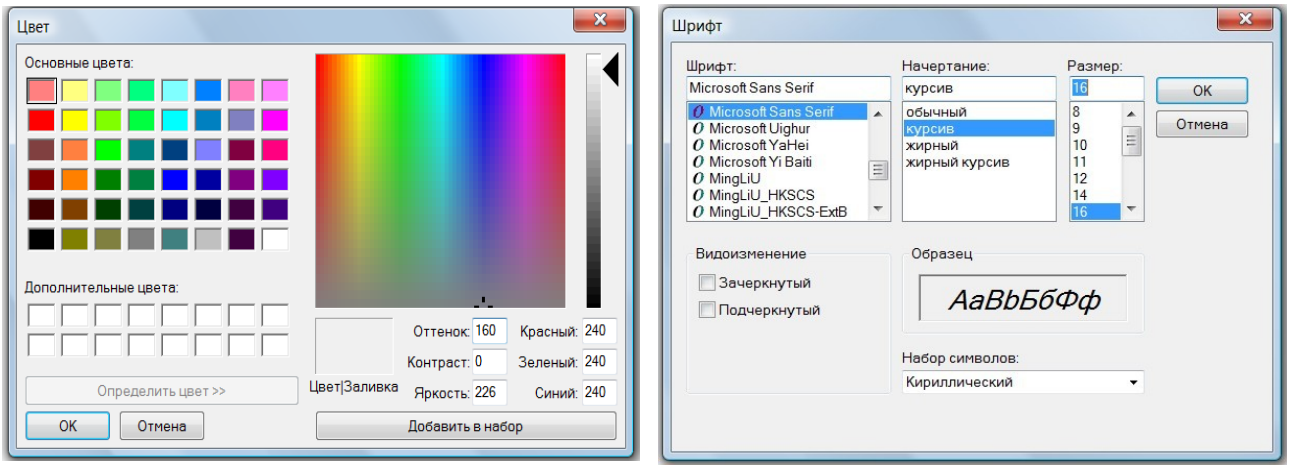


Рис. 3.19. Вид диалоговых окон **ColorDialog** (*а*) и **FontDialog** (*б*)

Диалоги **OpenFileDialog** (открытие файла) и **SaveFileDialog** (сохранение файла) возвращают дескрипторы выбранного файла, например:
string fn = openFileDialog1.FileName; или string fn = saveFileDialog1.FileName;

Пример 1

Создание простейшего текстового редактора. Шрифт и цвет текста изменяется с помощью меню.

1. Создадим новый проект **wf341** типа Windows Forms.

2. Разместим на форме размером 480×360 текстовое поле **textBox1** (свойства: **Multiline = true, ScrollBars = Vertical, Anchor** = Top, Bottom, Left, Right).

3. Перетащим на форму компоненты **ColorDialog, FontDialog** и **MenuStrip**.

4. Выделим компонент **MenuStrip.** Пользуясь подсказками, создадим два пункта меню: **вид** (с подпунктами **шрифт, цвет**) и **справка** (рис 3.20).

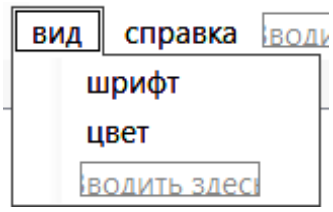


Рис. 3.20. Создание меню

5. Поочередно выделяем созданные пункты и регистрируем события **Click.**

6. В шаблоны обработчиков выбора пунктов меню **шрифт** и **цвет** введем коды, вызывающие диалоги задания шрифта и цвета текста:

```
private void шрифтToolStripMenuItem_Click(object sender, EventArgs e)
 { fontDialog1.ShowDialog();          // вызов диалога задания шрифта
    textBox1.Font = fontDialog1.Font;
 }
  private void цветToolStripMenuItem_Click(object sender, EventArgs e)
    { colorDialog1.ShowDialog();          // вызов диалога задания цвета
     textBox1.ForeColor = colorDialog1.Color;
 }
```

7. В шаблон обработчика выбора пункта меню **Справка** введем код вызова окна сообщений **MessageBox**:

```
private void справкаToolStripMenuItem_Click(object sender, EventArgs e)
 {  MessageBox.Show("текстовый редактор \n разработал: студент");
 }
```

8. Протестируем программу, вводя текст и изменяя цвет и шрифт. Резуль- тат может выглядеть так (рис. 3.21, *а*, *б*).

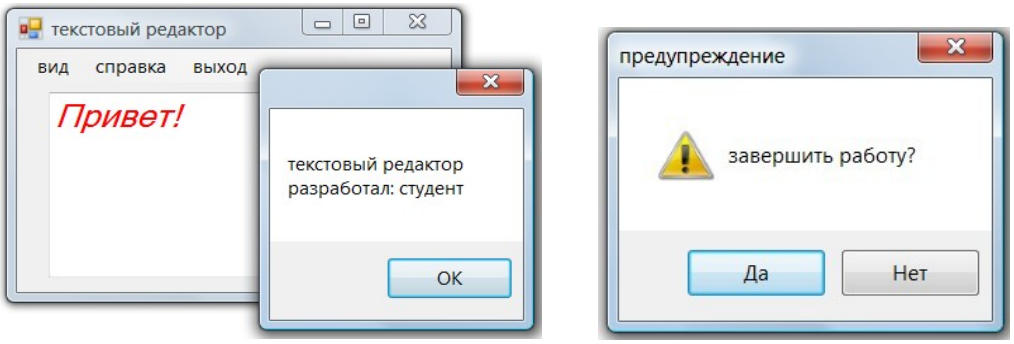


Рис. 3.21. Вид редактора (*а*) и окон сообщений Справка (*б*) и Выход (*в*)

9. Добавим пункт меню **выход**. Зарегистрируем событие **Click.** В шаблон обработчика введем код вызова окна сообщений с двумя кнопками (Да – Нет) и иконкой предупреждения (рис. 3.21, *в*):

```
private void выходToolStripMenuItem_Click(object sender, EventArgs e)
 {  DialogResult res = MessageBox.Show("завершить работу?", "предупреждение",
                       MessageBoxButtons.YesNo, MessageBoxIcon.Warning);
    if (res == DialogResult.Yes)  Application.Exit();
 }
```

10. Протестируем окончательный вариант.

Пример 2

Открытие и сохранение текстовых файлов.

1. Продолжим модифицировать простейший текстовый редактор (проект **wf341**). Добавим возможность открывать и сохранять текстовые файлы.

2. Для этого в начале файла с программным кодом **Form1.cs** подключим пространство имен **System.IO.**

3. Перетащим на форму компоненты: **openFileDialog** (открытие) и **SaveFileDialog** (сохранение файла).

4. Добавим в меню пункт **файл** с подпунктами **открыть** и **сохранить** и зарегистрируем для них события **Click.**

5. В шаблоны обработчиков введем коды вызова диалогов открытия и сохранения файлов:

```
private void открытьToolStripMenuItem_Click(object sender, EventArgs e)
 {  openFileDialog1.FileName = string.Empty;
     if (openFileDialog1.ShowDialog() == DialogResult.OK)
      {  string fn = openFileDialog1.FileName; this.Text
         = "открыт файл " + fn;
         try { StreamReader sr = new StreamReader(fn); textBox1.Text
                = sr.ReadToEnd();  sr.Close();
             }
        catch (Exception ex)
            {  MessageBox.Show("Ошибка чтения \n" + ex.ToString());  }
      }
 }
private void сохранитьToolStripMenuItem_Click(object sender, EventArgs e)
 {  if (saveFileDialog1.ShowDialog() == DialogResult.OK)
     {  string fn = saveFileDialog1.FileName; this.Text
        = "сохранен файл " + fn;
        if (fn != string.Empty)
         {  FileInfo fi = new FileInfo(fn);
            try { StreamWriter sw = fi.CreateText(); sw.Write(textBox1.Text);
                   sw.Close();
                }
            catch (Exception ex)
                { MessageBox.Show("Ошибка записи \n" + ex.ToString());  }
         }
     }
 }
```

6. Протестируем программу. Откроем текст из файла **stroki.txt** (рис. 3.22, *а*).

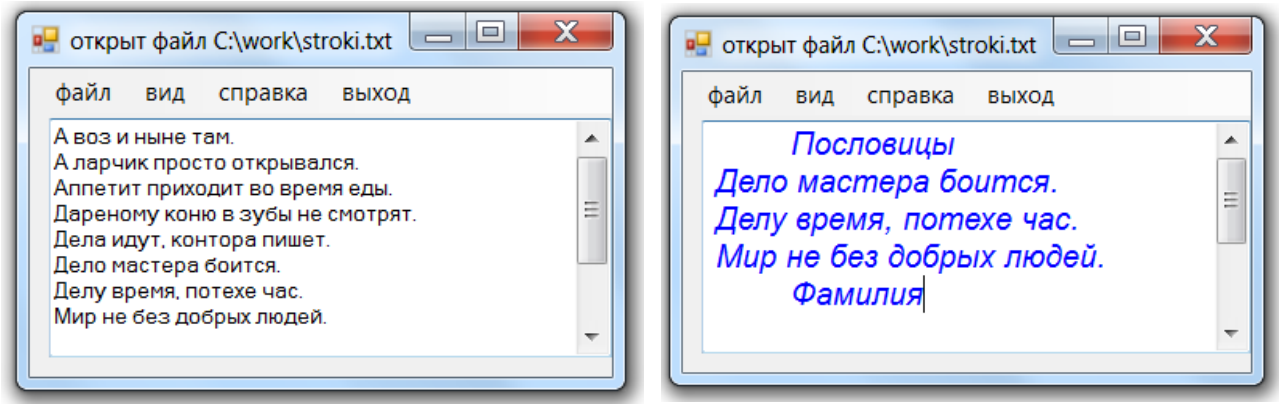


Рис. 3.22. Фрагмент текста до (*а*) и после редактирования (*б*)

7. Отредактируем и оформим текст: удалим первые 5 строк, вставим заго- ловок «Пословицы», а в конце свою фамилию; шрифт Arial, 12 пт, курсив, цвет синий (рис. 3.22, *б*). Сохраним файл под именем **stroki2.txt.**

Рабочее задание

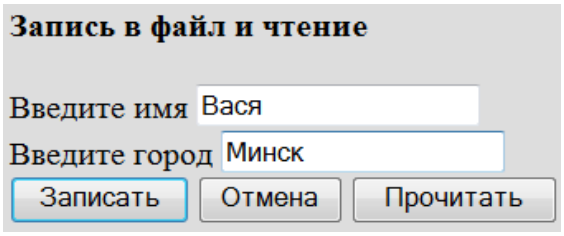
Выполнить инструкции приведенные ниже

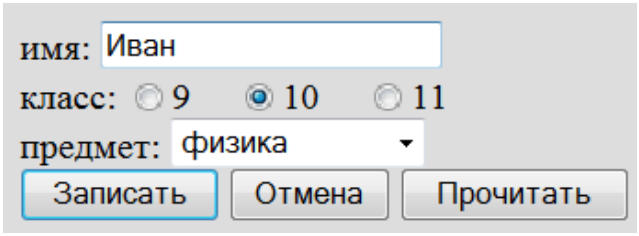
Порядок выполнения работы

1. Создайте приложение, в котором двойным щелчком мыши на форме вызывается компонент **colorDialog**, позволяющий изменять цвет формы.

2. Создайте приложение, в котором по нажатию кнопки вызывается ком- понент **openFileDialog**, позволяющий выбирать и загружать в **pictureBox** изоб- ражение из файла.

3. Создайте регистрационную форму, в которую вводится имя и город. Данные можно записывать в файл **sam3.txt** и читать по нажатию кнопок.





4. *Создайте регистрационную форму, в которой вводится имя, с помощью **radioButton** выбирается класс, из списка **comboBox** выбирается предмет (физи- ка, математика, информатика). Данные можно записывать в файл **reg.txt** и чи- тать из файла по нажатию кнопок.

5. *Создайте регистрационную форму, в которой в текстовые поля вводят **имя, логин, пароль** и **e-mail**. Корректность ввода проверяется с помощью ре- гулярных выражений. Если введены верные данные, они сохраняются в файле **log.txt**, иначе выводится сообщение «Повторите ввод».

Контрольные вопросы

1. Как создать главное меню в C# с использованием Windows Forms?
2. Как добавить подменю и пункты меню в главное меню в C#?
3. Как связать обработчики событий с пунктами меню в C#?
4. Как создать контекстное меню в C# для работы с элементами управления?
5. Как отобразить диалоговое окно сообщения с предупреждением или подтверждением в C#?
6. Как создать пользовательское диалоговое окно в C#?
7. Как передать данные из диалогового окна обратно в основное приложение в C#?
8. Как создать диалоговое окно выбора файла или папки в C#?
9. Как показать модальное диалоговое окно в C#?
10. Как добавить кастомные стили и темы к диалоговым окнам и меню в C#?

Практическая работа №19
Работа с графикой GDI+

Цель работы: формирование навыков работы с графикой в среде MS Visual Studio.

Теоретическая часть

Для работы с графикой в среде .NET предназначен класс **Graphics** про- странства имен **System.Drawing.** Для вывода графических примитивов (линий, геометрических фигур), текста, растровых изображений необходимо создать объект класса **Graphics**, например, методом **CreateGraphics**:

Graphics g = this.CreateGraphics();

После создания объекта типа **Graphics** можно применять его свойства и методы. Наиболее часто используются объекты и методы классов **Pen** (рисо- вание линий и геометрических фигур), **Brush** (заполнение областей), **Font** (ра- бота с текстом), **Color** (работа с цветом).

Для интерактивного управления свойствами графических объектов удоб- но использовать манипулятор мышь. События мыши **MouseDown, MouseUp, MouseMove** и другие работают в сочетании с делегатом **MouseEventHandler**.

Например, при регистрации события движения мыши по форме в методе **InitializeComponent()** (в файле Form1.Designer.cs) появляется строка: `MouseMove += new MouseEventHandler(Form1_MouseMove);`

При этом в файле кода Form1.cs создается шаблон метода-обработчика, которому передаются два параметра: объект-источник события и объект класса **MouseEventArgs**, который содержит информацию о событии, например: **X** и **Y** – координаты указателя мыши; **Button** – нажатая кнопка (левая, правая); **Clicks** – количество нажатий и отпусканий кнопки мыши; **Delta** – счетчик (со знаком) щелчков поворота колесика.

Эту информацию можно использовать в обработчике, например, для выво- да координат в заголовок формы:

```
public void Form1_MouseMove(object sender, MouseEventArgs e)
  {  Text = string.Format("координаты: x={0}, y={1}", e.X, e.Y);
  }
```

Для управления графическими объектами нередко используют и клавиату- ру. События клавиатуры **KeyUp, KeyDown** работают в сочетании с делегатом **KeyEventHandler.**

Например, при регистрации события нажатия клавиши записывается строка

```
KeyDown += new KeyEventHandler(Form1_KeyUp);
```

Создается шаблон метода-обработчика, которому передаются: объект-источ- ник события и объект класса **KeyEventArgs**, который содержит информацию о событии, например: **KeyCode** – код клавиши для событий KeyDown или KeyUp; **Modifiers** – какие модифицирующие клавиши (Shift, Alt, Control) были нажаты; **Handled** – было ли событие полностью обработано.

Эту информацию можно использовать в обработчике, например:

```
private void Form1_KeyDown (object sender, KeyEventArgs e)
  {  MessageBox.Show(e.KeyCode.ToString(), “клавиша нажата!");
  }
```

Пример 1

Вывод графических примитивов на форму.

1. Создадим новый проект **wf351** типа Windows Forms.
2. Подключим пространство имен **System.Drawing**.
3. Разместим на форме поле выбора со списком **listBox1**. В пункте **Item** ок- на свойств введем список графических примитивов: **Line, Rectangle, FillRec- tangle, Ellipse, FillEllipse, Pie, FillPie** (рис. 3.23).

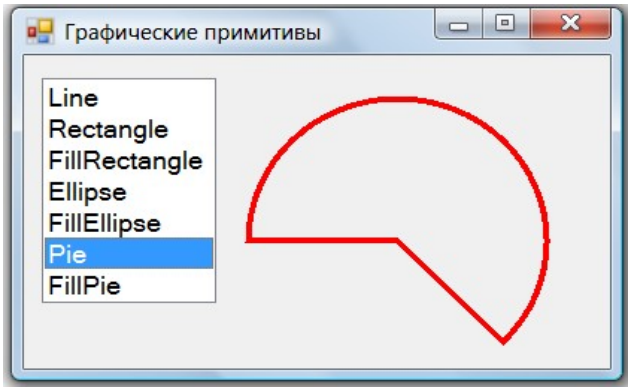


Рис. 3.23. Выбор и рисование графических примитивов на форме

4. Зарегистрируем событие выбора из списка **SelectedIndexChanged.** В шаб- лон обработчика введем код выбора и вывода примитивов на форму:

```
private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
```

```
{  Graphics g = this.CreateGraphics();  // создание графического объекта
   Pen pn = new Pen(Color.Red,4);       // создание пера
   Brush br = new SolidBrush(Color.Green); // создание кисти
   g.Clear(SystemColors.Control);       // очистка области цветом формы
   switch (listBox1.SelectedIndex)      // выбор и рисование примитива
    {  case 0: g.DrawLine(pn, 150,40, 350,180); break;
       case 1: g.DrawRectangle(pn, 150,30, 250,150); break;
       case 2: g.FillRectangle(br, 150, 30, 250, 150); break;
       case 3: g.DrawEllipse(pn, 150,30, 250,150); break;
       case 4: g.FillEllipse(br, 150, 30, 250, 150); break;
       case 5: g.DrawPie(pn, 150,30, 200,200, 180,225); break;
       case 6: g.FillPie(br, 150,30, 150,150, 0,45); break;
    }
}
```

5. Протестируем программу. Откорректируем свойства компонентов и код.

Пример 2

Простейший графический редактор. Рисование мышью.

1. Создадим новый проект **wf352** типа Windows Forms.

2. Разместим на форме кнопку для очистки (рис. 3.24, *а*).

3. Зарегистрируем три события мыши для формы (**MouseDown, MouseUp, MouseMove**) и событие нажатия кнопки **Click**. Введем коды в шаблоны обработчиков событий:

```
        // инициализация: перо поднято, цвет черный, толщина 4 px
bool ris = false; Color clr = Color.Black;  int w = 4;
        // обработчики событий мыши
private void Form1_MouseDown(object sender, MouseEventArgs e)
       {  ris = true;  }                                  // перо опущено
private void Form1_MouseUp(object sender, MouseEventArgs e)
       {  ris = false;  }                                 // перо поднято
                          // движение мыши, вывод ее координат в заголовок формы
private void Form1_MouseMove(object sender, MouseEventArgs e)
 {  this.Text = "x = " + e.X + "  y = " + e.Y;
    if (ris)                                             // если нажата кнопка мыши
     {  Graphics g = CreateGraphics();  // рисуем закрашенным квадратом
        g.FillRectangle(new SolidBrush(clr), e.X, e.Y, w, w);
     }
 }
private void button1_Click(object sender, EventArgs e)
 { Graphics g = CreateGraphics();
    g.Clear(SystemColors.Control);
 }
```

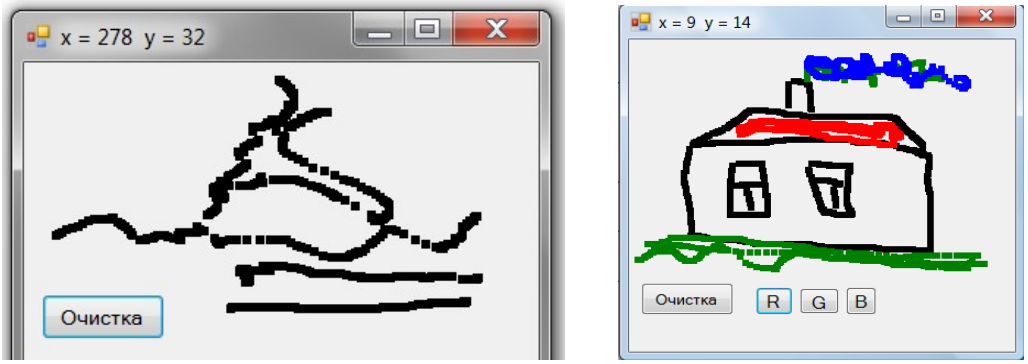


Рис. 3.24. Варианты интерфейса графического редактора

4. Протестируем программу. При необходимости откорректируем свойства элементов и код.

Пример 3

Вывод текста в графике.

1. Создадим новый проект **wf353** типа Windows Forms.

2. Разместим на форме надпись, поле ввода текста и три кнопки для выбора шрифта, цвета и вывода текста (рис. 3.25).

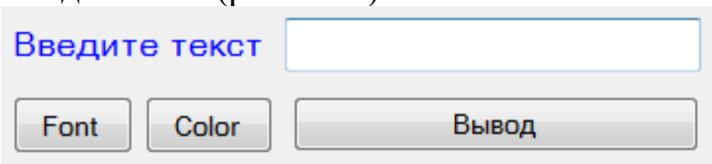


Рис. 3.25. Интерфейс приложения **wf353**

3. Зарегистрируем события нажатия кнопок. Введем код в шаблоны обработки:

```
        // инициализируем поля, задаем цвет текста и шрифт по умолчанию
Color clr = Color.Black;
Font fnt = new Font("Times New Roman", 14);
        // вызов диалога выбора шрифта
private void button1_Click(object sender, EventArgs e)
 { FontDialog fntDia = new FontDialog();
    fntDia.ShowDialog();   fnt = fntDia.Font;
 }
        // вызов диалога выбора цвета текста
private void button2_Click(object sender, EventArgs e)
 { ColorDialog colDia = new ColorDialog();
    colDia.ShowDialog();   clr = colDia.Color;
 }
        // ввод и рисование текста
private void button3_Click(object sender, EventArgs e)
 {  string s = textBox1.Text;                          // ввод текста
    Graphics   g   =   CreateGraphics();
    g.Clear(SystemColors.Control);  Brush
    br = new SolidBrush(clr);
    g.DrawString(s, fnt, br, 20, 100);                 // вывод текста
 }
```

4. Протестируем программу. При необходимости откорректируем свойства элементов и код.

Рабочее задание

Выполнить инструкции приведенные ниже

Порядок выполнения работы

1. Модифицируйте пример 1, добавив выбор цвета пера и кисти (Red, Green, Blue) с помощью **radioButton** и толщины пера с помощью **numericUpDown**.

2. Модифицируйте пример 2, добавив выбор цвета пера (Red, Green, Blue) с помощью кнопок и толщины пера с помощью клавиш **Up** и **Down**.

3. Создайте приложение, выводящее в указанное щелчком мыши место формы закрашенный кружок.

4. Модифицируйте пример 3, добавив вывод текста в указанное щелчком мыши место (событие MouseDown).

5. Создайте приложение, выводящее на форму **N** закрашенных квадратиков со случайными координатами

Контрольные вопросы

1. Какая технология используется для работы с графикой в C# в среде MS Visual Studio?
2. Как создать новый проект для работы с графикой в C# в MS Visual Studio?
3. Как добавить элемент управления для отображения графики в окно приложения в среде MS Visual Studio?
4. Как создать график с использованием элемента управления в C#?
5. Как добавить данные на график в C#?
6. Как изменить внешний вид графика (например, цвет линии, шрифт осей) в C#?
7. Как добавить метки на оси графика в C#?
8. Как добавить легенду к графику в C#?
9. Как сохранить график как изображение или печатать его в C#?
10. Как обновить и перерисовать график в реальном времени в C#?

Практическая работа №20

Создание класса и объекта. Методы. Конструкторы

Цель работы: формирование навыков создания класса, объекта, методов. Создание и использование конструкторов.

Теоретическая часть

Каждый объект реального мира обладает свойствами и поведением – набо- ром статических и динамических характеристик. Поведение объекта зависит от его состояния и внешних воздействий. Понятие объекта в программировании похоже на обыденный смысл этого слова. **Объект** – совокупность данных, характеризующих его состояние, и методов, моделирующих его поведение.

В объектно-ориентированном программировании (ООП) предметная об- ласть представляется как совокупность взаимодействующих объектов. Реализу- ется **событийная модель** взаимодействия: объекты обмениваются сообщения- ми и, реагируя на них, выполняют определенные действия.

Основные **принципы ООП**: абстрагирование, инкапсуляция, полиморфизм, наследование.

Абстрагирование – выделение существенных для данной задачи характе- ристик объекта и отбрасывание второстепенных. Любой программный объект – это абстракция. Детали реализации объекта, как правило, скрыты, они исполь- зуются через его интерфейс – совокупность правил доступа.

Инкапсуляция – сокрытие деталей реализации. Позволяет представить программу в укрупненном виде и защитить от нежелательных вмешательств.

Полиморфизм – использование одного имени (методов, операций, объек- тов) для решения нескольких схожих задач или для обращения к объектам раз- ного типа. Идея полиморфизма – «один интерфейс, множество методов». Воз- можны различные способы реализации полиморфизма: перегрузка методов, перегрузка операций, виртуальные методы, переопределение методов, параметри- зованные классы. Чаще всего понятие полиморфизма связывают с механизмом виртуальных методов.

Наследование – это процесс, посредством которого один объект может приобретать свойства другого. Для объекта можно определить потомков, кото- рые наследуют, корректируют или дополняют его поведение. Наследование дает возможность многократного использования программного кода.

Класс – обобщенное понятие, описывающее характеристики и поведение множества сходных объектов (называемых **экземплярами** или просто объекта- ми этого класса). В программе класс является пользовательским **типом данных** и представляет собой блок кода, в котором описывается одна сущность, на- пример, модель реального объекта или процесса. Основными элементами клас- са являются данные и методы их обработки.

Заголовок описания класса обязательно содержит служебное слово **class** и **Имя**, которое по правилам языка C# начинается с заглавной буквы. В теле класса в фигурных скобках **{ ... }** описываются его элементы. Тело может быть пустым.

[модификаторы] **class Имя** [: предки]
{ тело класса }

Перед заголовком класса могут указываться **модификаторы** (modifier), ко- торые определяют некоторые общие свойства класса, а также доступность для других элементов программы: **internal** – внутренний (задается по умолчанию, можно не писать); **public** – общедоступный; **private** – закрытый; **protected** – за- щищенный (закрыт для посторонних, но доступен для наследников); **static** – статический; **abstract** – абстрактный; **sealed** – запечатанный (или бесплодный, не может иметь наследников).

Класс может содержать следующие функциональные элементы *(members):*

– **поля** – переменные класса;

– **свойства** – обеспечивают «умный» доступ к полям;

– **методы** – определяют поведение класса;

– **конструкторы** – служат для инициализации объектов (экземпляров класса);

– **исключения** – используются для обработки исключительных ситуаций;

– набор **операций**, позволяющих производить различные действия. Простейший пример – описание общедоступного класса с одним методом:

```
public class  Computer
   {   public void ShowInfo()
          {   Console.WriteLine( "RAM =  8 Гбайт" );     }
   }
```

Все классы .NET имеют общего предка – класс **object**, и организованы в единую иерархическую структуру. Классы логически группируются в **прост- ранства имен**, которые служат для упорядочивания имен классов и предот- вращения конфликтов имен: в разных пространствах имена могут совпадать. Пространства имен могут быть вложенными. Любая программа использует пространство имен **System**, в котором собрано множество стандартных классов и методов (смотри раздел 1).

Рассмотрим теперь описание экземпляра класса. Напомним, что класс яв- ляется обобщенным понятием, определяющим характеристики и поведение мно- жества конкретных объектов, называемых **экземплярами** (или просто объекта- ми) этого класса. Классы создаются программистом до выполнения программы. Экземпляры класса (объекты) создаются системой во время выполнения. Про- граммист задает создание экземпляра класса с помощью инструкции **new**, на- пример: Computer comp1 = new Computer(“Asus”, 2048) или Computer comp2 =
= new Computer(“IBM”, 4096). При этом для каждого объекта выделяется от- дельная область памяти для хранения его данных. Класс может содержать так- же и статические элементы, которые существуют в единственном экземпляре.

Содержащиеся в классе данные могут быть переменными или константами. Описанные в классе переменные называются **полями класса**. При описании по- лей обязательно указывают **тип** и **имя** (которое начинается с малой буквы).

[модификаторы] **тип имя** [= начальное_значение]

Можно также указывать модификаторы, определяющие доступность, а так- же задавать начальное значение (т. е. инициализировать поле).

Все поля в C# сначала автоматически инициализируются нулем соответст- вующего типа. Например, полям типа int присваивается 0, полям типа double – 0.0, а ссылкам на объекты – значение null). После этого полю присваивается значе- ние, заданное при его явной инициализации.

Метод – именованный функциональный элемент класса, выполняющий вы- числения и другие действия. Метод определяет поведение класса. В программе метод представляет собой блок кода, содержащий ряд инструкций, к которому можно обратиться по имени. Он описывается один раз, а вызываться может многократно по необходимости.

При описании метода в заголовке обязательно указывают его **тип** (который соответствует типу возвращаемого методом значения) и **Имя** (с заглавной бук- вы). В круглых скобках после имени указывают параметры метода, которых может и не быть, но скобки обязательны.

[модификаторы] **тип Имя(** [параметры] **)**

{ тело метода }

В теле метода в фигурных скобках **{ ... }** описываются выполняемые этим методом действия. Тело метода может быть пустым.

Доступность и другие характеристики метода задаются модификаторами (смысл которых будет раскрываться по мере выполнения работ): **public, private, protected, internal, abstract, virtual, override, new, static, sealed.** Методы клас- са имеют доступ к его полям непосредственно или через свойства (**set – get**). Поскольку поля хранят данные, а методы выполняют действия, для облегчения чтения и понимания кода программы рекомендуется поля называть существи- тельными, а методы глаголами.

Параметры метода определяют множество значений аргументов, которые можно передавать в метод. Для каждого параметра обязательно задавать его **тип** и **имя**. Передаваемые в метод аргументы должны соответствовать объяв- ленным параметрам по количеству, типам и порядку. Имя метода вместе с количеством и типами его параметров составляет **сигнатуру** метода. В сигнатуру не входит тип возвращаемого методом значения. Методы различают благодаря сигнатурам. В классе не должно быть методов с одинаковыми сигнатурами.

В языке C# возможны четыре типа параметров: параметры-значения, па- раметры-ссылки (ref), выходные параметры (out), параметры-массивы (params).

При вызове метода сначала выделяется память под его параметры. Каждо- му из параметров сопоставляется соответствующий аргумент. Проверяется со- ответствие типов аргументов и параметров, и производятся необходимые пре- образования типов (или выдается сообщение о невозможности). После этого выполняются вычисления и/или другие действия (тело метода). В результате значение заявленного типа передается в точку вызова метода. Если методу за- дан тип **void**, он ничего не возвращает (т. е. является процедурой в терминоло- гии Pascal). После выполнения метода управление передается на выражение, следующее после его вызова.

Методы реализуют функционал класса. Хорошо спроектированный метод должен решать только одну задачу, а не все сразу. Необходимо четко представ- лять, какие параметры должен получать метод, и какие результаты выдавать. Необходимо стремиться к максимальному сокращению области действия каж- дой переменной. Это упрощает отладку программы, поскольку ограничивает область поиска ошибки.

Заметим, что элементы (поля, методы), характеризующие класс в целом, следует описывать как **статические**. Статический метод (с модификатором *static*) может обращаться только к статическим полям класса. Статический метод вы- зывается через имя класса, а обычный – через имя экземпляра.

Конструктор – особый вид метода, предназначенный для инициализации объекта (конструктор экземпляра) или класса (статический конструктор). Кон- структор объекта вызывается при создании экземпляра класса с помощью клю- чевого слова **new**. Имя конструктора **совпадает** с именем класса. Конструктор не имеет никакого типа, даже **void**.

Класс может иметь несколько конструкторов с разными параметрами для разных вариантов инициализации. Если не указано ни одного конструктора или некоторые поля не были инициализированы, полям значимых типов присваива- ется ноль (**0** или **0.0**), полям ссылочных типов – значение null. Конструктор, вы- зываемый без параметров, называется *конструктором по умолчанию*.

Пример 1

Создание класса и объекта. Создание и вызов метода. Конструкторы.

1. Создадим проект **con211**.

2. В едином пространстве имен **namespace con211** с шаблоном класса **Pro- gram** создадим класс **Build** (проект строения). По умолчанию он имеет моди- фикатор доступа **internal.**

3. В этом классе объявим три поля **name** (имя), **area** (площадь), **kvo** (количе- ство жильцов) с модификаторами доступа **public**. Создадим метод **ShowInfo()**, ко- торый вычисляет площадь на одного жильца и выводит информацию о строении.

```
class Build
{ public string name; public double area; public int kvo;
  public void ShowInfo()
  { Console.WriteLine("В доме {0} площадью {1} живет {2} чел, на человека {3:f2}",
                      name, area, kvo, area/kvo);
  }
}
```

4. В методе **Main()** класса **Program** создадим объект **dom1** класса **Build,** (т. е. построим дом по проекту **Build**), используя конструктор по умолчанию (без параметров). Зададим значения полей (параметры нашего дома). Вызовем метод **ShowInfo()**.

```
class Program
{ static void Main()
  {  Build dom1 = new Build();                  // создание объекта dom1
    dom1.name="Дача"; dom1.area=30; dom1.kvo=4;          // инициализация
    dom1.ShowInfo();                                     // вызов метода
    Console.ReadKey();
  }
}
```

5. Протестируем программу, изменяя параметры объекта **dom1**. Заметим, что при использовании конструктора по умолчанию значения полей задавать не совсем удобно.

6. Модифицируем программу. Создадим в классе **Build** собственный кон- структор с тремя параметрами (**nm, ar, k**). Конструктор по умолчанию (без па- раметров) теперь тоже необходимо записать в явном виде.

```
public Build() { }                              // конструктор без параметров
```

```
public Build(string nm, double ar, int k)   // конструктор с параметрами
  {  this.name = nm;   this.area = ar;  this.kvo = k;  }
```

Служебное слово **this** используют для указания на конкретный экземпляр объекта, а также для устранения неоднозначности между полями и локальными переменными или параметрами методов. Например, чтобы не плодить большое количество имен, полям (т. е. переменным класса) и используемым внутри ме- тода локальным переменным (значения которых передаются через соответст- вующие параметры), обычно дают одинаковые имена. При этом конструктор с параметрами будет выглядеть так:

```
public Build(string name, double area, int kvo)
   {  this.name = name;  this.area = area;  this.kvo = kvo;  }
```

Служебное слово **this** указывает на поля класса **Build** (выделены полужир- ным), которым при создании объекта будут присвоены значения локальных пе- ременных, переданные через одноименные параметры (выделены курсивом).

В дальнейшем мы тоже будем придерживаться такой практики задания имен.

7. В методе **Main()** класса **Program** создадим объект **dom2** класса **Build,** (т. е. построим новый дом по тому же проекту **Build**), используя теперь конст- руктор с параметрами. Снова вызовем метод **ShowInfo()**.

```
Build dom2 = new Build("Коттедж", 80, 6);
dom2.ShowInfo();
```

8. Протестируем программу, изменяя параметры объекта **dom2**. Заметим, что использование собственного конструктора гораздо удобнее.

Рабочее задание

Выполнить инструкции приведенные ниже

Порядок выполнения работы

1. Модифицируйте программу примера **1** (**con211**), добавив поле **floor** (ко- личество этажей). Создайте два объекта типа **Build** с разными параметрами и способами инициализации. Создайте приложения, в которых определяются классы, поля, конструкторы, методы, создаются и инициализируются 2–3 объекта.

2. Класс **Student**. Метод **ShowInfo** выводит фамилию, имя, курс, возраст.

3. Класс **Computer.** Метод **Info** выводит модель (IBM, Asus, Sony) и пара- метры компьютера (объем ОЗУ и жесткого диска).

4. Класс **Tovar**. Метод **Kupi** выводит название (тетрадь, книга, ручка), це- ну, наличие на складе (есть, нет), количество.

5. Класс **Pogoda**. Метод **Show** выводит город (Минск, Брест, Гомель), тем- пературу, осадки (ясно, пасмурно, гроза), направление и скорость ветра.

6. Класс **Transport**. Метод **ShowInfo** выводит параметры транспортного средства: тип (автомобиль, мотоцикл, велосипед), цвет, скорость, масса.

7. Класс **Animal**. Метод **Golos** выводит вид (кошка, собака, попугай), имя (Мурка, Шарик, Кеша), голос (мяу, гав, ррр).

8. Класс **Figura.** Метод **ShowArea** выводит название (квадрат, прямоуголь- ник) и параметры фигуры (основание, высоту), вычисляет и выводит площадь.

Контрольные вопросы

1. Что такое класс в C# и как он отличается от объекта?
2. Как объявить класс в C#?
3. Как создать объект класса в C#?
4. Как объявить метод внутри класса в C#?
5. Как вызвать метод объекта класса в C#?
6. Что такое конструктор в C# и зачем он нужен?
7. Как объявить конструктор внутри класса в C#?
8. Как использовать конструктор для инициализации полей объекта класса в C#?
9. Можно ли объявить несколько конструкторов внутри класса в C#? Как компилятор определит, какой конструктор вызывать?
10. Можно ли передавать параметры в конструктор класса в C#? Если да, то каким образом?

Практическая работа №21
Перегрузка методов

***Цель работы*:** формирование навыков создания перегруженных методов.

Теоретическая часть

Перегрузкой методов *(overloading)* называется использование методов с одним и тем же именем, но различным количеством и типами параметров. Компилятор по типу фактических параметров сам определяет, какой именно метод требуется вызвать. Это называется **разрешением** (*resolution*) перегрузки. Перегрузка методов – одна из простейших реализаций полиморфизма. Широко используется также перегрузка конструкторов.

Большинство стандартных операций тоже можно переопределять, что поз- воляет использовать объекты своих типов в составе выражений аналогично пе- ременным стандартных типов.

Пример 1

Использование перегрузки методов. Вычисление периметров разных фигур.

1. Создадим проект **con221**.

2. В классе **Program** создадим три статических метода с одинаковым именем **Perim**, но разными сигнатурами. В методе **Main** будем вызывать эти методы.

```
class Program
{
  static void Perim(int a, int b)                          // два параметра
  {  Console.WriteLine("Периметр прямоугольника = {0}", 2*a+2*b);
  }
  static void Perim(int a, int b, int d )                  // три параметра
  {  Console.WriteLine("Периметр треугольника = {0}", a+b+d);
  }
  static void Perim(params int[] ar)          // переменное число параметров
  { int p = 0; foreach (int x in ar) p += x; Console.WriteLine("Периметр
     {0}-угольника = {1}", ar.Length, p);
  }
  static void Main()
  {  Perim(10,20);    Perim(3,4,5);    Perim(2,3,4,5,6,7,9);
     Console.ReadKey();
  }
}
```

3. Протестируем программу, вызывая метод **Perim** с разными параметра- ми. Обратим внимание на вариант метода с переменным числом параметров.

Пример 2

Заказ номеров в гостинице. Использование перегрузки конструкторов.

1. Создадим проект **con222**.

2. В едином пространстве имен **con222** с классом **Program** создадим класс **Zakaz** с полями **fam** (фамилия), **size** (количество мест в номере), **comfort** (ком- фортность), методом **Show** (показать заказ) и четырьмя конструкторами с раз- ным числом и типами параметров.

```
class Zakaz
{
    private string fam; private int size; private string comfort;
        // создаем четыре конструктора
```

```
    public Zakaz(string fm, int sz, string cmf)          // 3 параметра
   {  fam = fm; size = sz; comfort = cmf;  }
   public Zakaz(string fm, int sz)                       // 2 параметра
   {  fam = fm; size = sz; comfort = "стандарт";  }
   public Zakaz(string fm)                               // 1-параметр
   {  fam = fm; size = 3; comfort = "стандарт";  }
   public Zakaz()                                        // без параметров
   {  fam = "неизвестный"; size = 6; comfort = "общежитие";  }
   public void Show()
   {  Console.WriteLine("{0} забронировал {1} местный номер класса {2}",
                         fam, size, comfort);   }    }
```

3. В методе **Main** класса **Program** будем создавать объекты класса **Zakaz** с разными параметрами и вызывать один и тот же метод **Show**.

```
class Program
{ static void Main()
   { Zakaz z1 = new Zakaz("Иванов", 1, "Люкс"); z1.Show(); Zakaz
      z2 = new Zakaz("Петров", 2);      z2.Show(); Zakaz z3 = new
      Zakaz("Сидоров");                          z3.Show();
      Zakaz z4 = new Zakaz();                    z4.Show();
      Console.ReadKey();  }
}
```

4. Протестируем программу, вызывая метод **Zakaz** с разными параметрами.

Рабочее задание

Выполнить инструкции приведенные ниже

Порядок выполнения работы

1. Модифицируйте программу примера 1 (**con221**), добавив вариант перегрузки метода **Perim** для определения периметра квадрата по его стороне: **4*a**.

2. Модифицируйте программу примера 2 (**con222**), добавив возможность многократного ввода заказа с клавиатуры (фамилия, количество мест в номере, комфорт). Завершение ввода – символ **Q**. Создайте приложения, в которых определяются классы, поля, конструкторы, методы, создаются и инициализи- руются 2–3 объекта.

3. Класс **Figura.** Метод **ShowArea** перегружен. В зависимости от количе- ства введенных параметров выводится название фигуры (один параметр – квад- рат, два – прямоугольник, три – трапеция), вычисляется и выводится площадь.

4. Класс **Tour**. Метод **TourCalc** перегружен. Стоимость тура вычисляется в зависимости от количества и типа введенных параметров: без параметров – Минское море, бесплатно; один параметр (страна) – 1 день, 50 руб; два пара- метра (страна, количество дней **n**) – стоимость = 50***n**.

Контрольные вопросы

1. Что такое перегрузка методов в C# и для чего она используется?
2. Как объявить перегруженный метод в C#?
3. Каким образом компилятор C# различает перегруженные методы?
4. Можно ли перегрузить методы только по типам параметров или можно использовать и другие критерии?
5. Какой тип возвращаемого значения может иметь перегруженный метод в C#?
6. Можно ли перегружать методы, меняя только типы параметров местами?
7. Можно ли основываться на типах параметров при перегрузке методов, имеющих различные модификаторы доступа?

8. Какой метод будет вызываться, если передать в перегруженный метод значимый тип, который точно соответствует двум перегруженным методам?
9. Можно ли в одном классе иметь перегруженные методы с одинаковыми именами, но разными модификаторами доступа?
10. Какое правило следует соблюдать при перегрузке методов, чтобы избежать неоднозначности?

Практическая работа №22
Инкапсуляция. Сокрытие полей, создание свойств

Цель работы: формирование навыков управления доступом к полям.

Теоретическая часть

Для защиты от нежелательных вмешательств доступ к полям и методам классов приходится закрывать или ограничивать (например, с помощью моди- фикаторов private, protected). Для организации управления доступом к полям класса служат **свойства** (properties). Как правило, свойство определяет методы доступа к закрытому полю и имеет следующий синтаксис:

```
[ модификаторы]  тип  Имя
        {
            {  get    код_доступа      }          // получение значения
            {  set    код_доступа      }          // установка значения
        }
```

В C# свойство имеет то же имя, что и соответствующее скрытое поле, только первая буква заглавная, например: **поле** private int **age**, свойство public int **Age.**

При обращении к свойству автоматически вызываются указанные в нем блоки чтения **get** и установки **set**. Может отсутствовать либо часть get, либо set, но не обе одновременно. Если отсутствует set, свойство доступно только для чтения (read only), если отсутствует get – только для записи (write only).

Пример 1

Сокрытие полей, создание свойств.

1. Создадим проект **con231**.

2. В едином пространстве имен **con231** с шаблоном класса **Program** созда- дим класс **Student**. В этом классе объявим два поля **fam** (фамилия) и **kurs** (курс). Создадим конструкторы и метод **ShowInfo()**, который выводит инфор- мацию о студенте.

Внимание! Для удобства отладки программы все поля и методы сначала объявляем **public** (общедоступные). Имена полей вводим с маленькой буквы!

```
class Student
 {
  public string fam;                          //поля сначала public
  public int kurs;
  public Student() { }                        // конструктор без параметров
  public Student(string fam, int kurs)        // конструктор с параметрами
    { this.fam = fam; this.kurs = kurs;       // fam и kurs сначала с малых букв
    }
  public void ShowInfo()                                  // метод ShowInfo
    { Console.WriteLine("Студент {0} курса {1}", kurs, fam);
    }
 }
```

Первоначальная общедоступность полей и методов класса **Student** позво- ляет при отладке программы в методе **Main()** класса **Program** создавать объек- ты класса **Student**, (т. е. описывать конкретных студентов по шаблону класса **Student**), а также вызывать метод **ShowInfo()**.

```
class Program
```

```
{ static void Main()
   {  Student  st1  =  new  Student("Иванов",  3);  st1.ShowInfo();
      Student  st2  =  new  Student("",  -7);  st2.ShowInfo();
      Console.ReadKey();
   }
}
```

3. Протестируем программу с разными параметрами. Существенный недостаток – незащищенность от ввода абсурдных данных (например, можно ввести, что студент st2 не имеет фамилии и учится на отрицательном курсе –7.

4. **Инкапсулируем** данные (скроем и защитим поля), создав свойства с методами **set** и **get** для управления доступом к полям. Система Visual Studio поз- воляет автоматизировать этот процесс.

5. Для этого устанавливаем курсор на **имя_поля** (например, **fam**), в меню **Рефакторинг** (Refactoring) выбираем пункт **Инкапсулировать поле** (Incapsulate field), в пункте **обновление ссылок** указываем **Все** и нажимаем **OK**. В появив- шемся диалоговом окно **Просмотр изменений ссылок**, показываются предлага- емые замены полей на свойства (как правило, это все методы и объекты, исполь- зующие значения полей, кроме конструкторов!). Соглашаемся с предложением, нажимая **Применить**. Доступ к полю **fam** будет изменен на **private** (закрытый) и методом **Fam** сгенерирован шаблон общедоступного **свойства** с именем **Fam** (с большой буквы), которое и будет использоваться теперь вместо поля **fam**.

6. Создадим свои правила доступа. Для этого будем вводить необходимый код в автоматически сгенерированные шаблоны **set** и **get**. Например, все фами- лии будем хранить большими буквами (преобразование зададим в блоке **set**). А если фамилия не введена (поле **fam** пустое), то блок **get** будет возвращать значение «неизвестный». Измененный фрагмент кода будет выглядеть так:

```
private string fam;                                  // поле стало закрытым
public string Fam                                    // сгененрировано свойство
{
  get  { return (fam != "") ? fam : "неизвестный"; }   // получение значения
  set { fam = value.ToUpper(); }                       // установка значения
}
```

7. Аналогичным образом инкапсулируем поле **kurs**. Защитим его от ввода и хранения абсурдных значений. Например, при вводе чисел <1 или >4 в поле **kurs** будет сохранено значение 0 (поступай на подготовительный курс!). `private int kurs;`

```
public int Kurs
{
  get {  return kurs; }
   set {  kurs = (value<1 || value>4) ? 0 : value;  }  // установка значения
}
```

8. На завершающем этапе заменим в конструкторе **имена скрываемых полей** (с малой буквы) на **имена открытых свойств** (с большой буквы):

```
public Student(string fam, int kurs)
  { this.Fam = fam; this.Kurs = kurs;
```

Автоматически такая замена в конструкторе не производится, поскольку окончательное решение о правах доступа должен принимать программист.

9. Протестируем окончательный вариант с разными параметрами.

Рабочее задание

Выполнить инструкции приведенные ниже

Порядок выполнения работы

1. Модифицируйте программу примера1 (**con231**).

2. Добавьте еще два поля: имя **name** и возраст **age**. Инкапсулируйте их, введя ограничения на возраст от 15 до 35 лет. Добавьте конструктор с четырьмя параметрами: **public Student(string fam, string name, int kurs, int age)**.

Протестируйте программу, изменяя параметры инициализации, например;

```
Student st3 = new Student("Петров", "Петр", -7, 120); st3.ShowInfo();
```

3. Создайте подсчет количества вызовов метода **ShowInfo** для каждого студента. Для этого добавьте поле **id** и задайте метод доступа к нему только для чтения (есть только **get**):

```
private int id = 300;
public int Id  {  get { return id++; }   }
```

Протестируйте окончательный вариант, повторяя инициализацию.

4. Создайте приложения, в которых определяются классы, поля, конструк- торы, создаются и инициализируются 2–3 объекта. Поля инкапсулируются. Информация выводится методом **Show.**

5. Создается класс **Avto** с полями: марка автомобиля **brand**, цвет **color**, ско- рость **skor**. Поля инкапсулируются с ограничениями (скорость от 20 до 120 км/ч).

6. Создается класс **Kadry** с полями: фамилия **fam**, возраст **age**, должность **dol**, стаж **staj**. Поля инкапсулируются с ограничениями (возраст от 16 до 60, стаж от 0 до 45).

7. Создается класс **Computer** с полями: модель **model,** объем ОЗУ **ram** и жесткого диска **hdd**. Поля инкапсулируются с ограничениями (объем ОЗУ от 2 до 32 Гбайт, жесткого диска от 200 до 2000 Гбайт.

8. ***Создается класс **Tovar** с полями: название **name**, цена **price**, количество **kvo**. Поля инкапсулируются с ограничениями (цена от 1 до 20, количество от 0 до 10). Вычисляется стоимость заказанного товара каждого вида и всего заказа.

Контрольные вопросы

1. Что такое доступ к полям класса в C# и зачем он нужен?
2. Какие уровни доступа можно задать для полей класса в C#?
3. Как объявить поле класса с доступом только внутри самого класса в C#?
4. Как объявить поле класса с доступом из любого места в текущей сборке в C#?
5. Как объявить поле класса с доступом из производных классов в C#?
6. Как объявить поле класса с доступом из любого места в программе в C#?
7. Какие модификаторы доступа можно использовать для полей класса в C#?
8. Что такое инкапсуляция в контексте доступа к полям класса в C#?
9. Какие принципы и практики следует соблюдать при управлении доступом к полям класса в C#?
10. Как выбрать подходящий уровень доступа к полям класса в конкретной ситуации?

Практическая работа №23
Визуальное проектирование классов

***Цель работы*:** формирование навыков визуального проектирования классов.

Теоретическая часть

Технологии визуального объектно-ориентированного проектирования про- грамм основаны на использовании принципов унифицированного языка моде- лирования **UML** (*Unified Modeling Language*), который представляет собой язык графического описания модели проектируемой системы.

Подобно тому, как алгоритмические конструкции представляются с помо- щью блок-схем, для изображения классов, а также связей между ними исполь- зуют **диаграммы классов** (*class diagrams*). Диаграмма классов представляет статическую модель системы, ее структуру. Она не описывает поведение сис- темы или механизмы взаимодействия экземпляров классов. Основная цель – показать классы, их состав и отношения между ними.

Основные элементы на диаграммах классов изображаются прямоугольни- ками, а их отношения – линиями. Стрелка, например, изображает наследование. Прямоугольник (класс) содержит секции, отражающие его состав. Следует под- черкнуть, что UML – это абстрактный язык описания и графического представ- ления системы. Его реализация в Microsoft Visual Studio использует привычные для программиста обозначения и термины. На рис. 2.1 приведен пример диа- граммы простейших классов в среде Visual Studio и описания класса Computer на языке C#.

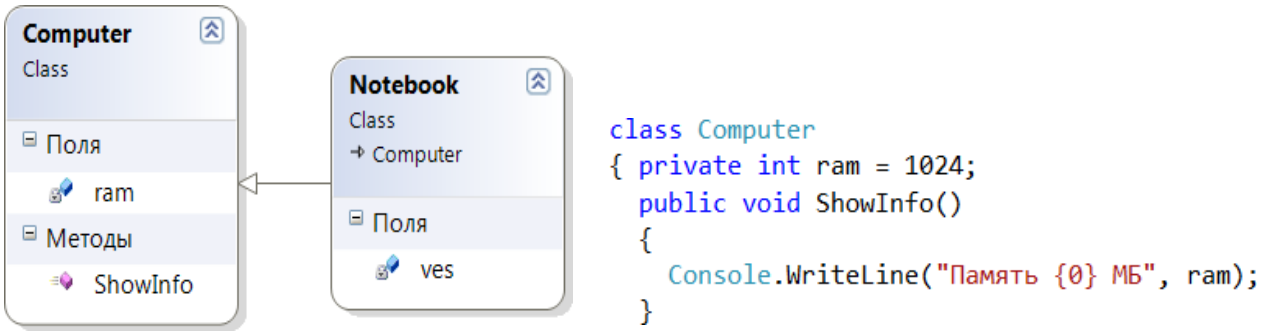


Рис. 2.1. Диаграмма классов и описание на языке C#

Система Visual Studio позволяет автоматизировать процесс проектирова- ния классов. Она содержит набор инструментов визуального конструирования классов в интерактивном режиме. Так, для создания и отображения классов разных типов предназначены графические шаблоны – стереотипы, которые от- личаются не только визуально, но и функционально (класс, абстрактный класс, интерфейс). Элементы класса (поля, свойства, методы) и их модификаторы по- мечены разными значками и отличаются по цвету.

Пример 1

Визуальное проектирование класса ***Computer***.

1. Спроектируем класс **Computer**, который содержит поля **model** (модель) и **ram** (оперативная память), а также методы **Start** (включение) и **End** (вы- ключение).

2. Создадим новый проект **con241**. По умолчанию он имеет класс **Program**.

3. В этом же пространстве имен в отдельном файле создадим класс **Com- puter** с помощью визуального **Конструктора классов** (Class Designer).

4. Для его вызова в окне **Обозреватель решений** (Solution Explorer) выде- лим **имя проекта** и щелкнем по значку **Перейти к схеме классов** (рис 2.2, *а*). Откроется окно **Схема классов** (ClassDiagrams) с созданным по умолчанию классом **Program**, содержащим шаблон метода **Main** (рис 2.2, *б*).

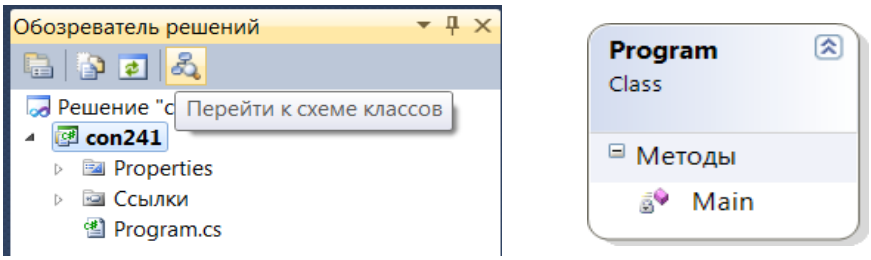


Рис 2.2. Окно **Обозреватель решений** (*а*) и изображение класса Program (*б*)

5. С помощью контекстного меню **Добавить** или **Панели элементов** доба- вим на схему класс **Computer**. Для этого на панели инструментов выберем пункт **Класс** (рис 2.3, *а*) и перетащим мышью в окно **Схема классов**. В от- крывшемся диалоговом окне **Новый класс** введем имя **Computer** (рис. 2.3, *б*). По умолчанию в нашем проекте будет создан новый файл **Computer.cs** с авто- матически сгенерированным шаблоном кода класса **Computer**.

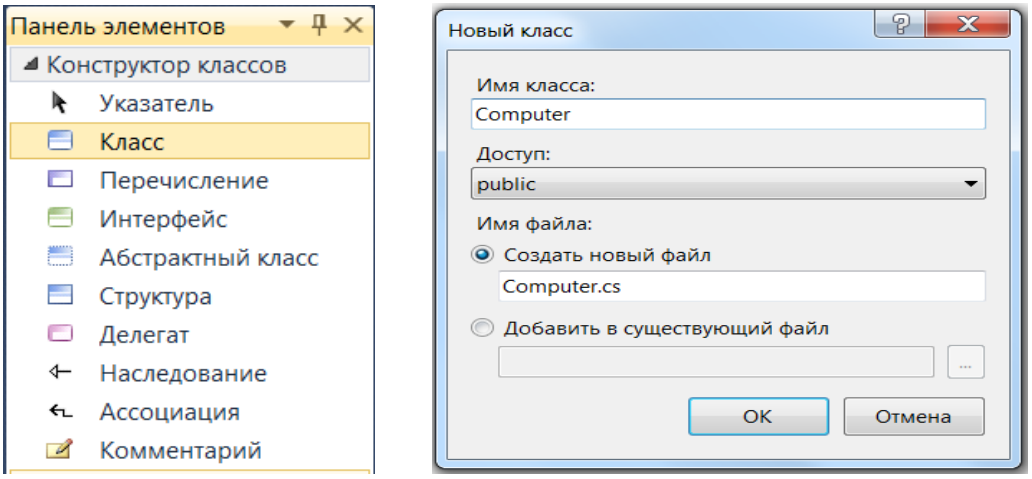


Рис. 2.3. Панель элементов (*а*) и диалоговое окно Новый класс (*б*)

6. Спроектируем теперь его элементы (поля, конструкторы, методы, свой- ства). Выделим на диаграмме изображение класса. Правой кнопкой мыши вы- зовем контекстное меню **Добавить** (рис. 2.4, *а*) и выберем требуемый элемент, например, **Поле**. Введем его имя **model** на изображении класса (рис. 2.4, *б*). Можно также воспользоваться панелью Свойства (рис. 2.4, *в*).

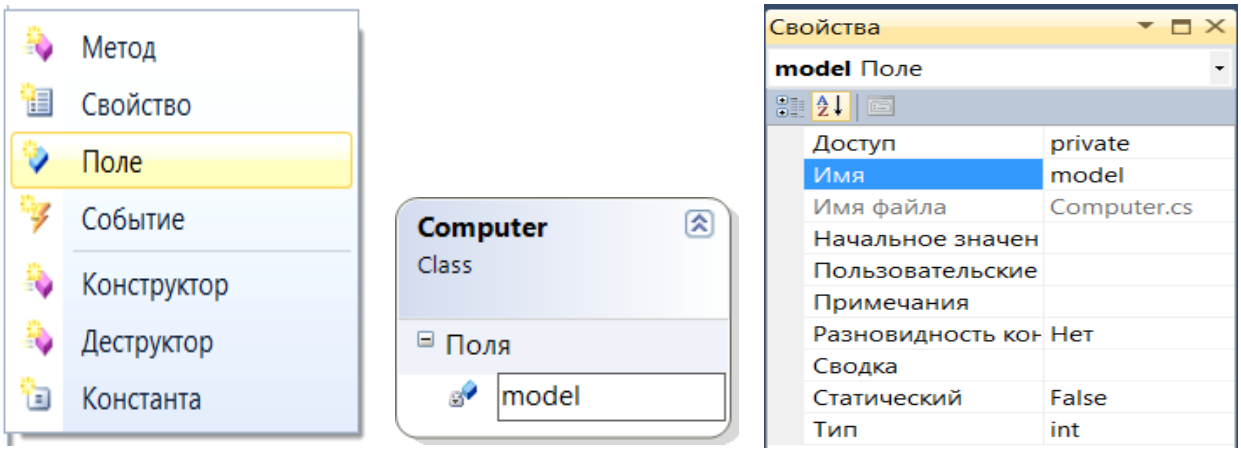


Рис. 2.4. Панель элементов (*а*), изображение класса (*б*) и панель Свойства (*в*)

7. На панели **Свойства** можно задавать характеристики каждого элемента. Однако удобнее использовать окно **Сведения о классах**, в котором в форме таблицы задаются имя, тип, модификатор и другие параметры сразу всех эле- ментов класса (рис. 2.5, *а*).

Будем создавать необходимые элементы класса **Computer** в соответствии с рис. 2.5, *б* и задавать их параметры в соответствии с рис. 2.5, *а*. Рекомендуем сначала задавать все поля, затем методы и конструкторы. Напомним, что в язы- ке C# конструктор является особым методом с тем же именем, что и класс (в нашем примере **Computer**), но не имеющим никакого типа (даже **void**)!

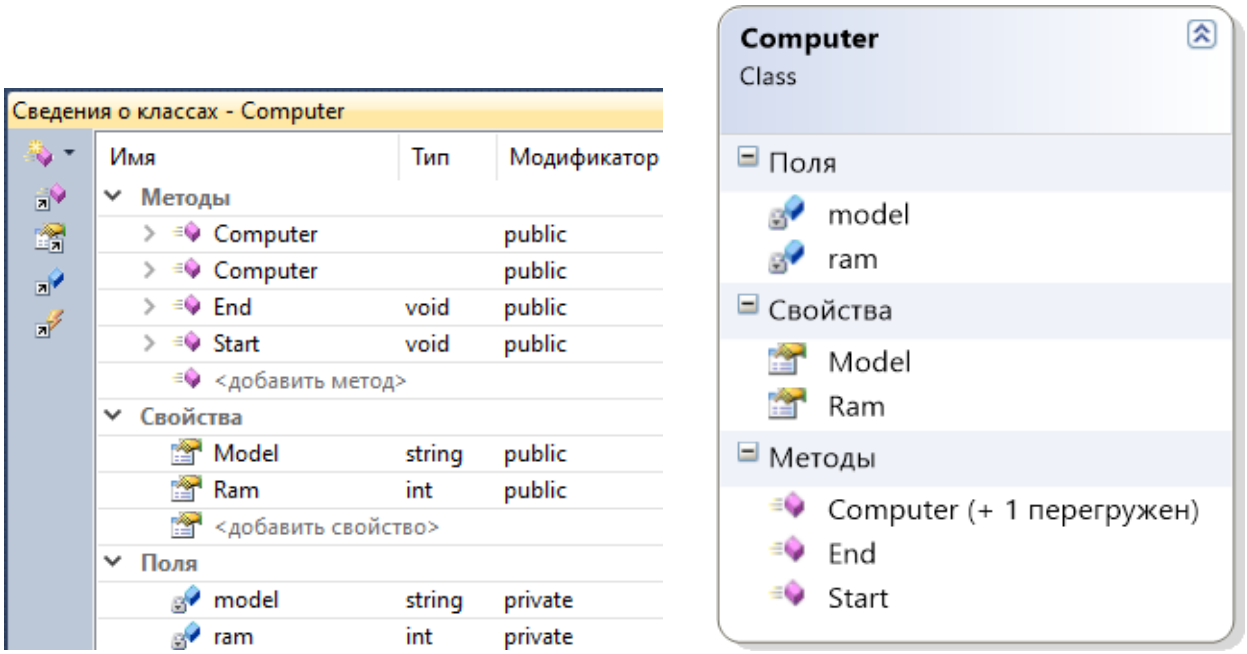


Рис. 2.5. Конечный вид окна **Сведения о классах** (*а*) и класса **Computer** (*б*)

В результате визуального проектирования в окне программы автоматически генерируются шаблоны элементов, в которые будем вводить программный код. На первом этапе целесообразно все поля задавать общедоступными (*public*), и лишь убедившись в отсутствии ошибок, инкапсулировать поля, настроив ограничения доступа в методах set – get. Окончательный вид программы:

```
public class Computer
{
  private string model; private int ram;          // скрытые поля
  public  string Model                            // общедоступные свойства
   {  get { return model ; }
      set { model = (value!="") ? value : "noName"; }
   }
  public int Ram
   {  get { return ram; }
      set { ram = (value < 500) ? 640 : value;  }
   }
  public Computer() {  }                    // конструктор без параметров
  public Computer(string model, int ram)                  // с параметрами
   {  this.Model = model;  this.Ram = ram;  }
  public void Start()                       // общедоступный метод Start
   {  Console.WriteLine("{0} работает, память = {1}", Model, Ram); }
  public void End()                         // общедоступный метод End
   {  Console.WriteLine("{0} выключается", Model); }
}
class Program
{  static void Main()
   { Computer comp = new Computer("IBM", 2048);
     comp.Start(); comp.End(); Console.ReadKey();
   }
}
```

9. Протестируем программу, изменяя параметры.

Порядок выполнения работы

Используя инструменты и методы визуального проектирования, создайте приложения, в которых определяются классы, поля, конструкторы, свойства. Информация выводится методом **Show.** В методе **Main** класса Program

созда- ются и инициализируются 2–3 объекта. Демонстрируется ограничение недопу- стимых значений.

Рабочее задание

1. Создается класс **Avto** с полями: модель автомобиля **model**, цвет **color**, ско- рость **skor**. Поля инкапсулируются с ограничениями (скорость от 60 до 150 км/ч).
2. Создается класс **Student** с полями: фамилия **fam**, возраст **age**, курс **kurs**. Поля инкапсулируются с ограничениями (возраст от 16 до 24, курс 2–5).
3. Создается класс **Sotrudnik** с полями: фамилия **fam**, стаж **staj**, зарплата **zar**. Поля инкапсулируются с ограничениями (стаж от 3 лет, зарплата от 500 руб).
4. Создается класс **Build** с полями: название **name**, площадь **area** (от 10 до 200), количество жильцов **kvo** (от 2 до 9). Поля инкапсулируются с огра- ничениями.
5. *Создается класс **Tovar** с полями: название **name**, цена **price**, количе- ство **kvo**. Поля инкапсулируются с ограничениями (цена от 1 до 10 руб, количе- ство от 0 до 10). Вычисляется стоимость заказанного товара.

Контрольные вопросы

1. Как создать новый проект в Microsoft Visual Studio для разработки классов на C#?
2. Как добавить новый класс в проект в Microsoft Visual Studio?
3. Как изменить имя класса в Microsoft Visual Studio?
4. Как добавить свойства (поля) в класс при визуальном проектировании в Microsoft Visual Studio?
5. Как добавить методы в класс при визуальном проектировании в Microsoft Visual Studio?
6. Как добавить конструктор в класс при визуальном проектировании в Microsoft Visual Studio?
7. Как создать наследование между классами в Microsoft Visual Studio?
8. Как добавить интерфейс к классу в Microsoft Visual Studio?
9. Как добавить комментарии к коду класса при визуальном проектировании в Microsoft Visual Studio?
10. Как использовать диаграмму классов в Microsoft Visual Studio для визуализации связей между классами?

Практическая работа №24
Наследование

Цель работы: формирование навыков реализации наследования.

Теоретическая часть

Наследование (inheritance) – это процесс приобретения состояния и пове- дения одного класса (называемого **базовым** или **предком**) другим классом (называемым **производным, наследником** или **потомком**). Для любого класса, кроме бесплодного (модификатор **sealed),** можно задать классы-наследники, в которых повторяется и дополняется состояние и поведение предка.

Синтаксис объявления производного класса (потомка):

[модификаторы] **class Имя : класс-предок, интерфейсы…**
{ тело класса }

Класс в C# может иметь произвольное количество потомков и только один класс-предок. В то же время, класс может наследоваться от произвольного ко- личества интерфейсов. Наследование позволяет многократно использовать про- граммный код, исключать из программы повторяющиеся фрагменты; упрощает модификацию программ и создание новых классов на основе существующих. Благодаря наследованию можно, например, использовать объекты, исходный код которых недоступен, но в поведение которых требуется внести изменения.

Наследование позволяет строить иерархии объектов. Они представляется в виде деревьев, в которых более общие объекты (предки) располагаются ближе к корню, а более специализированные (потомки) – на ветвях и листьях (рис. 2.6).

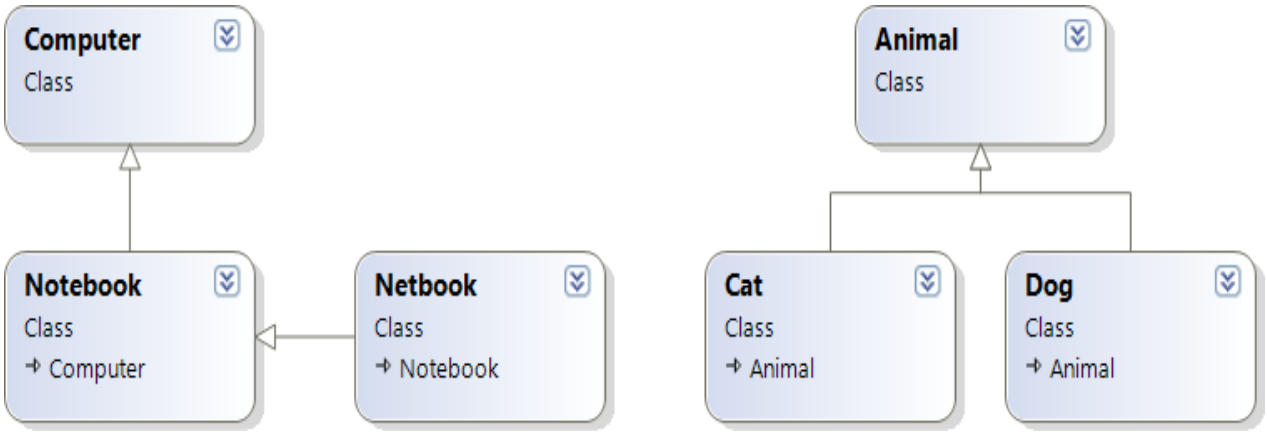


Рис. 2.6. Иерархии классов в окне Схема классов

Фрагменты программного кода для этих иерархий:

```
class Computer { ... }        // базовый класс        class Animal { ... }
class Notebook : Computer     // потомки              class Cat : Animal
  { ... }                                              { ... }
class Netbook : Notebook                               class Dog : Animal
        { ... }                                                { ... }
```

Возможны различные стратегии классического наследования:

– функционал потомка остается неизменным;

– функционал методов базового класса скрывается и замещается в производном класса (модификатор **new**);

– функционал методов базового класса, называемых **виртуальными** (модификатор **virtual**), переопределяется в производном классе (модификатор **override**).

Отметим важные особенности классического наследования:

– Наследуются поля, методы и свойства класса.

– Конструкторы не наследуются! Класс – потомок должен иметь собствен- ные конструкторы.

– Объекту базового класса можно присвоить объект производного, например:

```
public class Notebook : Computer  { … }
        Computer comp = new Notebook( … );
```

Таким образом, методы, которые у потомков должны реализовываться по- разному, при описании базовых классов следует определять виртуальными. Ес- ли во всех классах иерархии метод будет выполняться одинаково, его лучше определить как обычный метод. Виртуальные методы базового класса задают поведение всей иерархии, которое может изменяться и дополняться в потомках за счет добавления новых виртуальных методов. С помощью виртуальных ме- тодов реализуется один из основных принципов ООП – полиморфизм.

Отметим, что элементы базового класса с модификатором **private** в классе- наследнике недоступны, для них следует использовать модификатор **protected**, а для самого базового класса **public** или **internal.**

Спроектируем класс **Notebook**, который имеет два поля **model** (модель) и **ram** (оперативная память), а также методы **Start** (включение) и **End** (выключение). Ранее мы уже создали класс **Computer** с такими полями и методами. Имеет смысл считать класс **Notebook** потомком класса **Computer**, который наследует эти поля и метод **Start** без изменения (при включении они работают одинаково). Особен- ность ноутбука – наличие батареи, которая может заряжаться и после выключения ноутбука. Поэтому в класс **Notebook** надо добавить поле **time** (требуемое время зарядки батареи) и переопределить метод **End**, чтобы он показывал это время.

Пример 1

Реализация наследования.

1. Откроем ранее созданный проект **con241** с классами **Program** и **Computer**.

2. С помощью визуального конструктора классов в том же пространстве имен в отдельном файле создадим класс **Notebook.**

3. Для задания отношения наследник-предок на **Панели элементов** выбе- рем пункт **Наследование** и мышью с нажатой левой кнопкой протащим стрел- ку от

прямоугольника класса **Notebook** до класса **Computer** (рис 2.7).

4. С помощью окна **Сведения о классах** зададим поля и методы класса **Notebook** в соответствии с рис. 2.7.

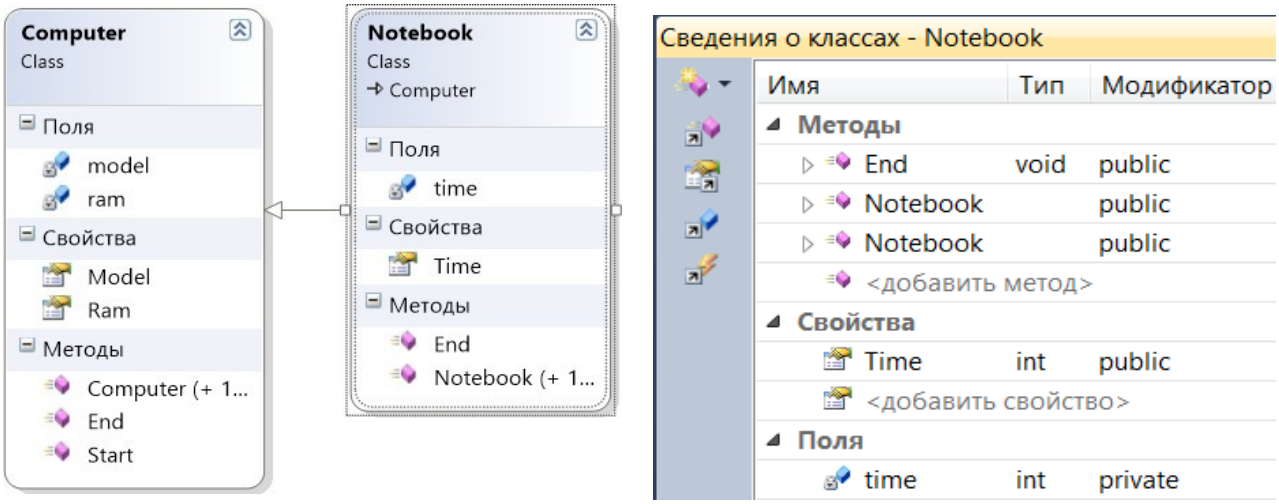


Рис. 2.7. Проектирование класса Notebook

5. Введем в автоматически сгенерированные шаблоны элементов класса **Notebook** программный код. Инкапсулируем поле **time.**

```
public class Notebook : Computer
{
  private int time;
  public int Time
   {  get { return time;  }
      set { time = (value < 10) ? 15 : value;  }    }
   // Конструкторы не наследуются, а вызываются! Поэтому их нужно создавать
   // в каждом классе-потомке. При этом можно ссылаться на базовый класс
   // и лишь добавлять новые параметры.
  public Notebook() { }
  public Notebook(string model, int ram,  int time) :  base(model,ram)
     {  this.Time = time;  }
  // Создадим новый метод End класса – наследника с модификатором override.
  public override void End()
  {  Console.WriteLine("{0} выключается, заряд {1} мин", Model, Time);
  }
}
  // Чтобы он смог переопределить одноименный метод базового класса,
  // зададим тому в классе – предке Computer модификатор virtual.
```

6. Протестируем программу, добавив в класс **Program** создание экземпляров класса **Notebook** и вызовы методов:

```
Notebook nb = new Notebook("Asus", 1024, 120);
   nb.Start();      nb.End();
Computer comp2 = new Notebook("Dell", 4096, 30);
   comp2.Start();  comp2.End();
```

7. Обратим внимание на важное проявление принципа полиморфизма: объект **comp2** имеет тип **Computer**, но ведет себя как ноутбук, поскольку ини- циализируется с помощью конструктора **Notebook.**

Рабочее задание

Выполнить инструкции приведенные ниже

Порядок выполнения работы

1. Модифицируйте пример **1 (con241):** добавив **public class Netbook** – по- томок **Notebook**. В нем определите поле **mas** (масса). Переопределите метод **Start**, добавив

информацию о массе. Протестируйте, изменяя параметры.

2. Используя инструменты и методы визуального проектирования, создай- те классы с наследниками, содержащие указанные поля, конструкторы и мето- ды. В классе **Program** создаются и инициализируются 2–3 объекта и указанны- ми методами выводится информация. Продемонстрируйте результаты, изменяя параметры.

3. Спроектируйте класс **Transport** с полями **model** (модель)**, speed** (ско- рость), **mas** (масса) и методами **Start**, **Stop**, **ShowInfo.** Наследуйте от него клас- сы **Avto** (автомобиль), **Moto** (мотоцикл), **Velo** (велосипед).

4. Спроектируйте класс **Animal** с полями **name** (имя)**, ves** (вес)**, col** (цвет) и методами **Run** (бегать), **Sleep**(спать), **Golos** (голос). Наследуйте от него клас- сы **Cat** (Кот) и **Dog** (Собака). Переопределите метод **Golos** для каждого живот- ного (**мяу-мяу** и **гав-гав**).

5. Спроектируйте класс **Tovar** с полями **name** (название), **price** (цена) и ме- тодом **Calc** (расчет и печать стоимости). Наследуйте от него классы **Book** (книги) с полем **kvo** (количество), **Pen** (ручки) с полем **kvo** и **Candy** (конфеты) с полем **ves** (вес). Переопределите метод **Calc** для расчета стоимости каждого товара.

Контрольные вопросы

1. Что такое наследование в C# и для чего оно используется?
2. Как объявить класс-наследник в C# и какие ключевые слова следует использовать?
3. Как унаследовать методы и свойства от базового класса в класс-наследнике в C#?
4. Можно ли унаследовать несколько классов одновременно в C#? Если нет, то почему?
5. Какие основные принципы наследования следует соблюдать при разработке классов в C#?
6. Что такое конструкторы в классах-наследниках и как они взаимодействуют с базовым классом в C#?
7. Как переопределить метод родительского класса в классе-наследнике в C#?
8. Что такое ключевое слово "base" в C# и как оно используется при наследовании?
9. Как обращаться к членам базового класса из класса-наследника в C#?
10. Какие могут быть проблемы с наследованием в C# и как их избежать?

Практическая работа №25
Абстрактные классы. Интерфейсы

Цель работы: формирование навыков создания абстрактных классов и интерфейсов.

Теоретическая часть

Абстрактные (abstract) классы предназначены для представления общих понятий, которые предполагается конкретизировать в производных классах. Абстрактный класс задает общее поведение для всей иерархии, при этом его методы могут не выполнять никаких действий. Такие методы называются аб- страктными, они имеют пустое тело и объявляются с модификатором **abstract.**

Абстрактный метод – это виртуальный метод без реализации. Абстракт- ный метод должен быть переопределен в любом неабстрактном производном классе. Если в классе есть хотя бы один абстрактный метод, весь класс также должен быть объявлен как абстрактный. Абстрактный класс может содержать и полностью определенные методы (в отличие от интерфейса).

Абстрактный класс не разрешает создавать свои экземпляры. Он служит только для порождения потомков. Как правило, в нем лишь объявляются мето- ды, которые каждый из потомков будет реализовывать по-своему. Это бывает полезным, если использовать переопределение методов базового класса в по- томках затруднительно или неэффективно. Например, имеет смысл объявить класс **Tovar** абстрактным. В нем задать общие для всех товаров поля: **name** (название), **price** (цена), **summa** (стоимость). А вот конкретные характерис- тики для каждого вида товаров и соответствующих им классов-потомков будут разные: для класса **Book** (книги) это количество экземпляров (поле **kvo**), для класса **Candy** (конфеты) – это граммы

(поле **ves**), для класса **Tkany** (ткани) – это метры (поле **dlina**). Метод **Calc** (расчет) будет для каждого товара свой. Поэтому в базовом классе его надо объявить абстрактным (из-за чего и класс **Tovar** приходится делать абстрактным), а у каждого потомка реализовать по- своему. В то же время базовый абстрактный класс может содержать и обычные методы, например **PrintSum** (печать суммы), который наследуются всеми потомками без изменений.

Интерфейс – крайний случай абстрактного класса, у которого нет полей и ни один метод не реализован. В нем объявляются только абстрактные методы, которые должны быть реализованы в производных классах.

Синтаксис интерфейса аналогичен синтаксису класса:

[модификаторы] **interface Имя** [**: предки**]
{ тело_интерфейса }

Тело интерфейса составляют только абстрактные методы, шаблоны свойств, а также события. Элементы интерфейса по умолчанию общедоступны. Интер- фейс не может иметь обычных методов – все элементы интерфейса должны быть абстрактными.

В языке C# разрешено одиночное наследование для классов и множествен- ное – для интерфейсов. Это позволяет придать производному классу свойства нескольких интерфейсов, реализуя их по-своему. Сигнатуры методов в интер- фейсе и реализации должны полностью совпадать. Для реализуемых элементов интерфейса в классе следует указывать спецификатор public. К этим элементам можно обращаться как через объект класса, так и через объект типа соответ- ствующего интерфейса.

Интерфейс можно представлять как «контракт» о реализации объявленных методов потомками. Таким образом, наследование интерфейса заключается в его **реализации** (implementation) потомками. Основная идея использования интерфейса состоит в том, чтобы к объектам разных классов можно было об- ращаться одинаковым образом. Каждый класс может определять элементы ин- терфейса по-своему. Так реализуется полиморфизм: объекты разных классов по-разному реагируют на вызовы одного и того же метода. Заметим, что в биб- лиотеке классов **.NET** определено множество стандартных интерфейсов, декла- рирующих желаемое поведение объектов.

Пример 1

Реализация абстрактных классов и интерфейсов.

1. Создадим новый проект **con241**.

2. С помощью визуального конструктора классов спроектируем в отдель- ном файле абстрактный класс **Avto**, и в этом же файле классы-потомки **Vaz** и **Maz**, а также интерфейсы **ITurbo** и **IEco** в соответствии с рис. 2.8.

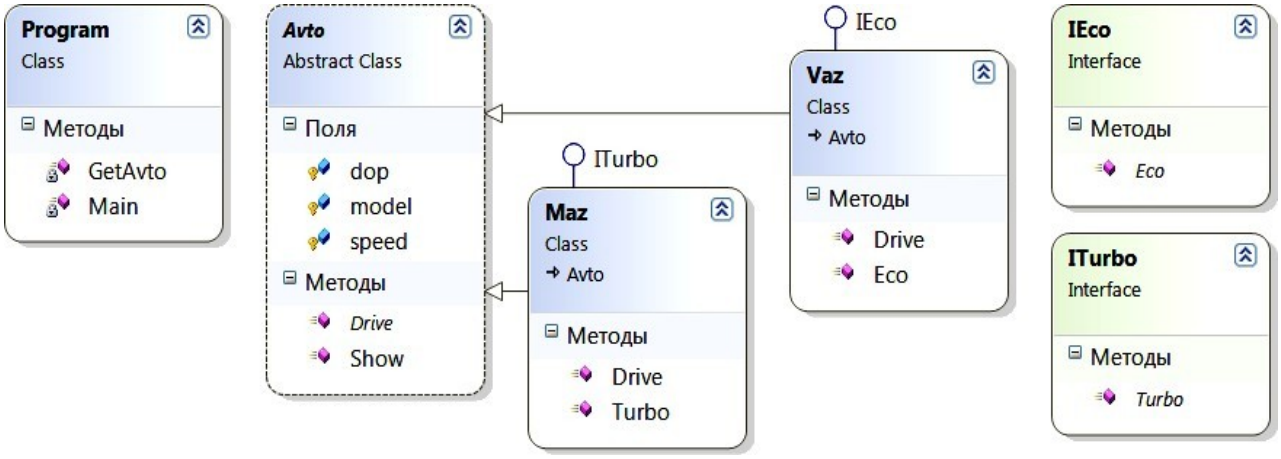


Рис. 2.8. Диаграмма классов

3. Введем в сгенерированные автоматически шаблоны программный код:

```
public abstract class Avto
{
 // объявляем поля с модификатором protected, доступные в классах-потомках
  protected string model; protected int speed;
  protected string dop;
  public abstract void Drive();          // объявляем абстрактный метод Drive
```

```
  public void Show()                    // объявляем обычный метод Show
   {  Console.WriteLine("модель {0}, скорость {1}, двигатель {2}",
                        model, speed, dop);    }
}
interface ITurbo  {  void Turbo();  }          // объявляем интерфейсы
interface IEco    {  void Eco();    }
            // класс Maz  наследует класс Avto и реализует интерфейс ITurbo
public class Maz : Avto, ITurbo
{ public override void Drive()     // переопределяем метод Drive
     { model = "Maz200"; speed = 90; Turbo(); }
  public void Turbo()  {  dop="турбо";  }     // реализуем интерфейс Turbo
}
            // класс Vaz  наследует класс Avto и реализует интерфейс IEco
public class Vaz : Avto, IEco
{ public override void Drive()// переопределяем  метод Drive
     { model = "vaz2107";  speed = 70; Eco(); }
  public void Eco() {  dop = "экологичный"; }   // реализуем интерфейс Eco
}
class Program
{
  static void Main()
  {  Avto myAvto = GetAvto();                  // создание объекта myAvto
    myAvto.Drive(); myAvto.Show();             // вызов методов
    Console.ReadKey();
  }
  static Avto GetAvto()                        // метод выбора автомобиля
  { Console.Write("Введите марку автомобиля (Vaz, Maz): "); string
     mod = Console.ReadLine();
     switch (mod)
      { case "Vaz": return new Vaz(); case
         "Maz": return new Maz(); default:
         return new Maz();
      }
  }
}
```

4. Протестируем программу, изменяя параметры.

Рабочее задание

Выполнить инструкции приведенные ниже

Порядок выполнения работы

1. Модифицируйте проект **con241**, добавив еще один класс-потомок **BMV** (интерфейсы **Turbo, Eco**) и параметры автомобилей (мощность двигателя, рас- ход топлива). Используя инструменты и методы визуального проектирования, создайте классы с потомками, содержащие указанные поля, конструкторы и ме- тоды. В классе **Program** создаются и инициализируются 2–3 объекта и указан- ными методами выводится информация. Продемонстрируйте результаты, изме- няя параметры.

2. Спроектируйте абстрактный класс **Animal** с полями (**name, rost, ves**) и классы-потомки **Cats** и **Dogs**. Абстрактный метод **Golos** переопределяется для каждого вида (мяу-мяу и гав-гав).

3. Спроектируйте абстрактный класс **Figura** и классы–потомки: **Rectangle** (прямоугольник) и **Circle** (круг). Абстрактный метод вычисления площади **Area**

переопределяется для каждой фигуры.

4. Спроектируйте абстрактный класс **Tovar** и классы-потомки: **Obuv** (обувь), **Odejda** (одежда), **Posuda** (посуда). Задайте общие (название, цена) и особенные (количество, размер, масса, цвет) поля и свойства. В классе **Tovar** задайте абстрактный метод **CalcSum** (расчет стоимости) и обычный **Print** (вывод всей информации). Абстрактный метод **CalcSum** переопределяется для каждого товара.

Контрольные вопросы

1. Что такое абстрактный класс в C# и для чего он используется?
2. Как объявить абстрактный класс в C# и какие ключевые слова следует использовать?
3. Можно ли создать экземпляр абстрактного класса в C#? Почему?
4. Какая разница между абстрактным классом и интерфейсом в C#?
5. Как объявить интерфейс в C# и какие ключевые слова следует использовать?
6. Можно ли унаследовать абстрактный класс от другого абстрактного класса в C#?
7. Какие могут быть члены абстрактного класса в C#?
8. Можно ли реализовать несколько интерфейсов в C# в одном классе? Если да, то как?
9. Какие основные преимущества использования интерфейсов в C#?
10. Когда следует использовать абстрактные классы, а когда интерфейсы в C#? Какое решение выбрать в конкретной ситуации?

Практическая работа №26
СТОИМОСТНАЯ ОЦЕНКА ПРОЕКТА

Цель работы: научиться определять стоимостную оценку проекта и определить сроки окупаемости внедряемой ИС при указанных затратах на проект внедрения.

Порядок выполнения работы:

−выполнить задание;
−показать преподавателю;
−ответить на вопросы преподавателя.

Время выполнения: 2 ч

Теоретическая часть:

Разработанная система предназначается для использования заместителем директора школы по АХЧ муниципального образовательного учреждения «Северодвинская общеобразовательная школа №23».

Основными задачами стоимостной оценки являются расчет периода окупаемости затрат на внедрение, разработку автоматизированной системы и оценка экономического эффекта, ожидаемого в результате внедрения разработанного проекта.

Результатом внедрения автоматизированной системы может быть прямой или косвенный экономический эффект. Под прямым экономическим эффектом понимается положительный результат от внедрения, выраженный в получении прибыли (реальных денег), чаще достигаемый за счет фактического сокращения персонала, т. к. комплекс работ, выполняемый на ЭВМ, позволяет отказаться от дополнительного увеличения штата сотрудников.

Косвенный (или условный) экономический эффект - это эффект, получаемый за счет внедрения программных средств, которые изменяют содержание работы персонала в сторону оперативности и надежности, но не приводят к обязательному фактическому сокращению штата сотрудников.

Может также наблюдаться социальный эффект (т.е. как скажется внедрение автоматизированной системы на работу заместителя директора школы по АХЧ с персоналом, с организациями – подрядчиками и т.д.) и характерный для автоматизированных систем технологический эффект (т.е. снижение трудоемкости за счет ускорения процедуры поиска необходимой информации, повышение качества принимаемых решений за счет автоматизации расчетов, производимых ранее вручную).

В данной методике оценки представлены следующие расчеты:

1) Расчет трудоемкости реализации функций до и после внедрения ИС;
2) Расчет затрат на реализацию функций ИС;

3) Расчет затрат на проектирование и внедрение системы

4) Расчет периода окупаемости и затрат на внедрение и разработку ИС.

Внедрение данного проекта даст возможность значительно снизить трудозатраты и существенно повысить качество работы.

Показателем экономической эффективности при внедрении автоматизированного рабочего места заместителя директора школы по административно-хозяйственной части является годовой экономический эффект и срок окупаемости (обратный показатель).

Годовой экономический эффект от использования адаптированной системы

определяется по формуле: $Э = (З1-З2)- \Delta К*ЕН$, (5.1)

З1, З2 – текущие затраты на обработку экономической информации по базовому и новому вариантам, руб.;
ЕН – нормативный коэффициент эффективности капитальных вложений.

е К – капитальные дополнительные вложения предприятия, связанные с внедрением результатов дипломного проекта, руб.

Сначала рассчитываем трудоемкость реализации функций до внедрения и после внедрения информационной системы. Расчет трудоемкости реализации функций до внедрения информационной системы сведен в таблице 5.1

Таблица 5.1 - «Стоимость работ до эксплуатации ИС»

Шаг процесса (функция)	Трудоемкость (чел-мин)	Повторяемость в год	Трудоемкость в год (чел-мин)	Трудоемкость в год (чел-дней)
Внесение данных о поступивших/ выданных материальных ценностях в книгу складского учета (1 строка)	0,5	630	315	0,7875
Регистрация товарных накладных	0,5	65	32,5	0,08125
Внесение данных о поступивших материальных ценностях (спецодежда)вжурналучета спецодежды	0,5	10	5	0,0125
Составление ежемесячной ведомости	60	12	720	1,8

выдачи МЦ на нужды учреждения				
Заполнение раздела инвентарной карточки ОС "Индивидуальная характеристика"	25	15	375	0,9375
Заполнение раздела инвентарной карточки ОС "Сведения о перемещениях"	20	20	400	1
Заполнение раздела инвентарной карточки ОС "Ремонт/модернизация"	20	140	2800	7
Внесение данных материальных ценностей в паспорт помещения	5	30	150	0,375
Составление общей заявки на закупку материальных ценностей от сотрудников	40	1	40	0,1
Составление акта на списание мягкого и хозяйственного инвентаря	20	24	480	1,2
Составление актов на списание основных средств	40	10	400	1
Составление акта о бое, ломе посуды	20	12	240	0,6
Составление списка фактических остатков по категории МЦ	90	4	360	0,9
Составление списка фактических остатков по всем МЦ, находящихся в учреждении	120	1	120	0,3
Внесение записи о прохождении инструктажа сотрудником школы	0,2	120	24	0,06
Выписка сотрудников, не прошедших инструктаж (по различным причинам)	30	2	60	0,15
Заполнение/корректировка карты аттестации рабочего места	30	5	150	0,375
Составление списка сотрудников для прохождения медосмотра	0,5	1	0,5	0,00125
Составление графика осмотра помещений	30	6	180	0,45
Запись в журнале работ о задании для сотрудника в период каникул	5	60	300	0,75
Отслеживание выполнения работы персоналом (с приведением испол. МЦ)	20	60	1200	3
Отслеживание договоров, заключенных с подрядчиками и актов выполненных работ	15	20	300	0,75
Итого в человеко-днях				21,63

Примечание: При расчете трудоемкости выполнения функций в год в человеко-часах следует исходить из нормативной продолжительности рабочего дня 480 минут

♣ перерывов в работе по 10 минут после каждых 50-ти минут работы на ЭВМ (указанные нормативы перерывов предусмотрены действующими санитарно-гигиеническими нормами). Таким образом, время реальной работы человека в день составляет 400 минут.

Все функции приведенные в таблице 5.1 выполняет заместитель директора школы по АХЧ, что составляет 21,63 ч/д в год. Эти функции являются только небольшой частью его должностных обязанностей – они связаны, в основном, с материально-техническим обеспечение учебного процесса. Кроме этих функций в обязанности заместителя директора школы по АХЧ входит: текущий контроль за санитарно-гигиеническим состоянием здания, газовой котельной, сооружений, классов, учебных кабинетов, мастерских, спортивной площадки и школьного стадиона, иного имущества школы, а также буфета в соответствии с требованиями

норм и правил безопасности жизнедеятельности; поиск и заключениепредварительных договоров с организациями, которые могут поставлять МЦ или предоставлять услуги более качественно или дешевле, контролирует выполнение обязательств со стороны организаций коммунального хозяйства, руководит работами по благоустройству, озеленению и уборке территории школы (летом); ведет учет рабочего времени технического и обслуживающего персонала школы и т.д.

Расчет трудоемкости реализации функций усовершенствованной информационной системы приведен в таблице 5.2.

Таблица 5.2 - Расчет трудоемкости реализации функций усовершенствованной ИС

Шаг процесса (функция)	Трудоемкость (чел-мин)	Повторяемость в год	Трудоемкость в год (чел-мин)	Трудоемкость в год (чел-дней)
Запись справочной информации в БД	0,1	30	3	0,0075
Внесение данных о поступивших/ выданных материальных ценностях в базу данных	0,1	630	63	0,1575
Регистрация товарных накладных	0,1	65	6,5	0,01625
Внесение данных о поступивших материальных ценностях (спецодежда) в базу данных	0,1	10	1	0,0025
Составление ежемесячной ведомости выдачи МЦ на нужды учреждения	0,2	12	2,4	0,006
Заполнение раздела инвентарной карточки ОС "Индивидуальная характеристика" в БД	5	15	75	0,1875
Заполнение раздела инвентарной карточки ОС "Сведения о перемещениях" БД	2	20	40	0,1
Заполнение раздела инвентарной карточки ОС "Ремонт/модернизация" БД	2	140	280	0,7
Внесение данных материальных ценностей в паспорт помещения и в БД	10	30	300	0,75
Ввод данных заявок от сотрудников в БД	0,2	20	4	0,01
Составление общей заявки на закупку материальных ценностей от сотрудников	0,2	1	0,2	0,0005

Составление акта на списание мягкого и хозяйственного инвентаря	20	24	480	1,2
Составление актов на списание основных средств	5	10	50	0,125
Составление акта о бое, ломе посуды	5	12	60	0,15
Составление списка фактических остатков по категории МЦ	0,2	4	0,8	0,002
Составление списка фактических остатков по всем МЦ, находящихся в учреждении	0,2	1	0,2	0,0005
Внесение записи о прохождении инструктажа сотрудником школы	0,1	120	12	0,03
Выписка сотрудников, не прошедших инструктаж (по различным причинам)	0,2	2	0,4	0,001
Заполнение/корректировка карты аттестации рабочего места	30	5	150	0,375
Составление списка сотрудников для прохождения медосмотра	0,1	1	0,1	0,00025
Составление графика осмотра помещений	15	6	90	0,225
Запись в журнале работ о задании для сотрудника в период каникул ивБД	5,5	60	330	0,825
Отслеживание выполнения работы персоналом	3	60	180	0,45
Отслеживание договоров, заключенных с подрядчиками и актов выполненных работ	0,5	20	10	0,025
Итого в человеко-днях				5,339

Проанализировав таблицы 5.1 и 5.2, уже можно сделать вывод о том, что годовая трудоемкость выполнения функций уменьшится за счет снижения времени, затрачиваемого на составление производных отчетов (в основном, за счет автоматизации расчетов остатков, но и за счет других отчетов: список сотрудников, не прошедших инструктаж, на медосмотр, общей заявки на МЦ) и таких документов, ведомость выдачи материальных ценностей на нужды учреждения, актов списания (разного типа), т.к. эти документы формируются на основании ранее внесенных данных практически автоматически, а ведомость полностью автоматизирована.

Таким образом, автоматизация некоторых процессов позволит заместителю директора школа по АХЧ более быстро решать поставленные перед ним задачи. Например, решения о закупках материальных ценностей, остатки которых приближаются к критическим, или которые необходимы сейчас для обеспечения непрерывности учебного процесса, получения мгновенной информации о запрашиваемом бухгалтерией основном средстве, решения о списании основных средств вследствие разных причин и т.д.

При расчете трудоемкости реализации функций после внедрения автоматизированной системы учитывается ввод справочной информации в систему, которой не будет хватать после первоначальной загрузки данных.

На следующем этапе необходимо произвести расчет затрат на реализацию функций спроектированной системы, исходя из трудозатрат до и после внедрения системы, а так же из среднего количества рабочих дней в месяце (22) и среднего оклада заместителя директора школы по АХЧ - 8250**.**

Расчет затрат на реализацию функций системы представлен в таблице 5.3.

Таблица 5.3 - «Расчет затрат на реализацию функций ИС»

Статья затрат	Затраты до внедрения (руб.)	Затраты после внедрения (руб.)
Расходы по оплате труда, в т.ч.	19061,44	4704,99
-Основная з/пл	8111,25	2002,13
-Дополнительная з/пл, в т.ч.	10950,19	2702,87
Льготы крайнего севера 80%	6489,00	1601,70
Районный коэффициент 40%	3244,50	800,85
Вознаграждение за работу (стимулирующие надбавки)15%	1216,69	300,32
Начисления на заработную плату 30% к сумме расходов на оплату труда (статья 1)	5718,43	1411,50
ИТОГО текущих затрат ($З_1$ и $З_2$ соответственно):	24779,87	6116,49

Начисления на заработную плату в 2013 году составляют 30%, и, по сути, являются страховыми взносами во внебюджетные фонды. Эта сумма распределяется между фондами, согласно установленного процентного соотношения:

1. в Пенсионный фонд Российской Федерации уплачивается 22 процента;
2. в Фонд социального страхования Российской Федерации - 2,9 процента;

3. в Федеральный фонд обязательного медицинского страхования - 3,1 процента;

4. в территориальные фонды обязательного медицинского страхования - 2,0 процента.

Согласно ч. 1 ст. 58.2 Закона N 212-ФЗ данный тариф применяется в течение 2012 - 2013 гг. В 2012 г. взносы в указанном размере перечисляются за каждого работника на суммы выплат, не превышающие 512 000 руб. (Постановление Правительства РФ от 24.11.2011 N 974), которая исчисляется нарастающим итогом с начала календарного года (ч. 4 ст. 8 Закона N 212-ФЗ).
Путем калькуляции затрат, направляемых на внедрение системы, рассчитываем
1.таблице 5.4 дополнительные капитальные вложения учреждения (ΔК).

Таблица 5.4 - «Расчет дополнительных затрат предприятия»

№ п/п	Наименование статьи	Сумма (руб)
1	Затраты на приобретение и монтаж оборудования, всего в т.ч.	0
2	Затраты на программное обеспечение	0
3	Затраты на оплату времени работы вычислительных и коммуникационных ресурсов, всего в т.ч.	7 400
	машинное время	6400
	время работы в Internet	1 000
4	Затраты на обучение персонала, всего в т.ч.	3750
5	Расходы по оплате труда проектировщика в т.ч.	30 500
	Основная заработная плата	12 200

	Дополнительная заработная плата (120%), в т.ч.	
	- с учетом льгот Крайнего севера(80%)	
	- с учетом районного коэффициента (40%)	14 640
	Начисления на заработную плату (30%)	3660
	ИТОГО:	41 650

Примечание:

и Затраты на приобретение и монтаж оборудования, на программное обеспечение нулевые, так как в сентябре 2012 года в кабинете директора школа по АХЧ был установлен новый компьютер (стоимость составила 29850 руб.) с достаточно высокими эксплуатационными характеристиками и со всем необходимым программным обеспечением согласно приобретенной лицензии.

и Затраты на оплату времени вычислительных ресурсов складываются из фактически затраченного времени на разработку проекта: 2 месяца (4 недели в каждом) * 5 дней в неделю * 8 часов = 320 часов и стоимости часа машинного времени 20 рублей, а так же времени работы в сети Internet – 40 часов по 25 рублей (Интернет использовался, в основном, для разработки аналитического раздела проекта, используется расценки Интернет кафе).

и Затраты на обучение персонала определяются исходя из того, что стоимость часа обучения на компьютере составляет 250 руб. Время, требуемое для обучения заместителя директора школы по АХЧ для работы с информационной системой составляет пять дней по 3 часа.

4, Расходы на оплату труда проектировщика. Основная заработная плата проектировщика за 2 месяца составит 30500 руб. (по окладу 6100 руб. плюс все начисления).
После произведения расчетов:

$З_1$ = 24779,87 $З_2$ = 6116,49, $\Delta К$ = 41650, ЕН равный 0,15, т.е. 15% (данные получены с помощью справочной системы «Консультант Плюс»).

Подставив все полученные значения в формулу 5.1, получим сумму годового экономического эффекта от внедрения адаптированной системы:

Э = (24779,87 -6116,49) –41650* 0,15 = 12415,88 (руб.)

Так как не планируется дополнительно закупка технического и программного обеспечения, то сумма ежегодной амортизации не рассчитывается.

Для расчета периода окупаемости нужно вычислить показатель - индекс доходности, который рассчитывается по формуле:

индекс доходности = ДП / ИС (5.2),

где ДП – сумма денежного потока (в настоящей стоимости) за весь период эксплуатации программного продукта (до начала инвестиций);
ИС - сумма инвестиционных средств, направляемых на реализацию проекта;

В качестве сумм денежного потока может служить показатель годового экономического эффекта, приведенного к настоящей стоимости путем дисконтирования.

Если значение индекса доходности меньше или равно 1, проект не принесет дополнительного дохода инвестору в текущем году и только когда индекс доходности становится больше 1, с этого периода начинает поступать доход от инвестиций.

Доход от внедрения АС это накопленный годовой экономический эффект за минусом остаточной суммы затрат, это абсолютное выражение сумм дохода или убытков, получаемых предприятием в i –м году эксплуатации АС.

Расчет дохода от внедрения системы представлен в таблице 5.5.

Таблица 5.5 - «Расчет дохода от внедрения системы»

Год эксплуатации ИС	**1**	**2**	**3**	**4**	**5**
Остаточная сумма затрат (тыс. руб.)	41,65	41,65	41,65	41,65	41,65
Сумма ежегодной амортизации на оборудование и ПО	0,00	0,00	0,00	0,00	0,00
Накопленный годовой экономический эффект	12,42	46,82	81,22	115,62	150,02
Индекс доходности	29,81%	112,40%	195,00%	277,59%	360,18%
Доход от внедрения ИТУ	-29,23	5,17	39,57	73,97	108,37

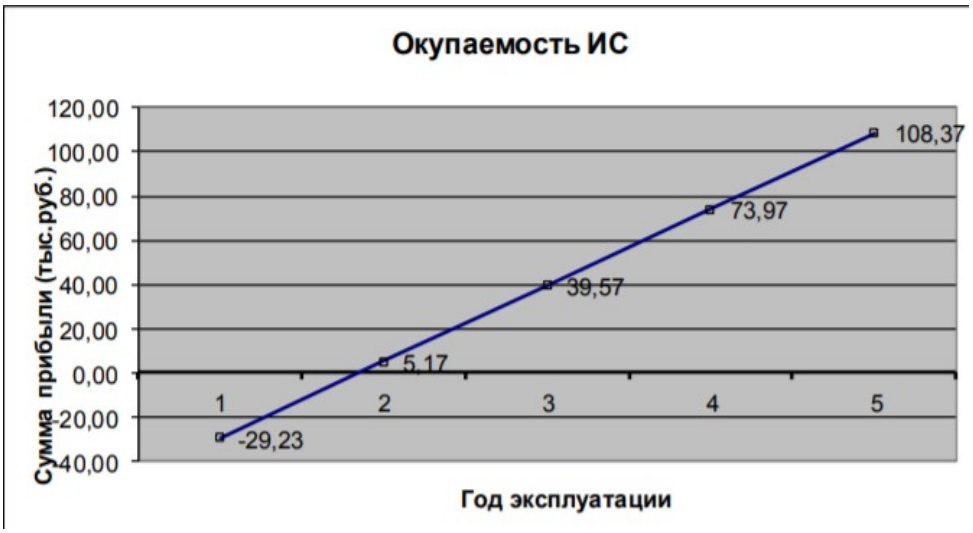


Рис.5.1 - График окупаемости затрат

Исходя из расчетов, система окупится на второй год эксплуатации. При финансово-экономических расчетах, связанных с инвестированием средств, процессы наращения и дисконтирования стоимости могут осуществляться как по формуле простых, так и по формуле сложных процентов. Поскольку инвестирование данного проекта менее трех лет, для расчета суммы дисконта применяется формула простых процентов.

Расчет дисконтированных сумм по простым процентам осуществляется по формуле:

$D=Э/(1+n*i)$ (5.3)

где Э – годовой экономический эффект;

в – продолжительность инвестирования (год эксплуатации);

i – фиксированная ставка рефинансирования, равная 8,25% (величина ставки получена на сайте Центра экономического анализа и экспертизы. Сайт - http://www.assessor.ru/forum/index.php?t=1599).

Расчет суммы дисконта по простым процентам представлен в таблице 5.6.

Таблица 5.6 - «Сумма дисконта по простым процентам»

Год эксплуатации ИС	1	2	3	4	5	6
Остаточная сумма затрат	41,65	41,65	41,65	41,65	41,65	35,90

Сумма ежегодной амортизации	0,00	0,00	0,00	0,00	0,00	0,00
Дисконтированная сумма ежегодной амортизации	0,00	0,00	0,00	0,00	0,00	0,00
Сумма денежного потока	12,42	12,42	12,42	12,42	12,42	12,42
Дисконтированная сумма денежного потока	11,47	12,41	12,42	12,42	12,42	12,42
Накопленная дисконтированная сумма денежного потока	11,47	23,88	36,30	48,72	61,14	73,56
Индекс доходности	27,54%	57,33%	87,15%	116,97%	146,79%	204,90%
Доход от внедрения ИС	-30,18	-17,77	-5,35	7,07	19,49	37,66

График окупаемости затрат представлен на рисунке 2.

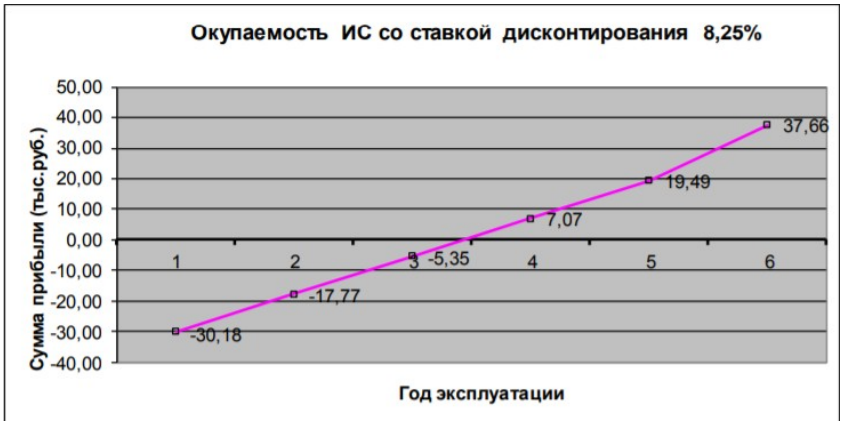


Рисунок 2 - График окупаемости ИС с учетом дисконтирования простым процентом

Таким образом, годовой экономический эффект от внедрения системы (Э), исходя из расчетов составил 12415,88 рублей. Автоматизированная система окупается на второй год эксплуатации (без приведения денежного потока к настоящей стоимости) и на третий год при расчете настоящей стоимости по формуле простых процентов.

Полученный экономический эффект является косвенным, так как увольнение персонала в школе не предполагается, и эффект выражается не конкретной суммой прибыли, а снижением трудоемкости выполнения рабочих функций.

Технологический эффект от внедрения разрабатываемой системы состоит в заметном сокращении времени на формирование выходных документов: ведомости выдачи МЦ на нужды учреждения, актов списания, инвентарной карточки, списков. Заместителю директора школы по АХЧ не придется затрачивать значительное время на формирования списков остатков для разных категорий МЦ и общего списка остатков МЦ в учреждении – всё это результативно сделает автоматизированная система с отсутствием ошибок на основе заранее введенных данных. Данные вводятся в систему через экранные формы (приложение Е). для быстроты и удобства пользователя применяются маски ввода для дат на всех таких полях. При формировании текущих документов даты их составления или ввода записей в табличные части этих документов проставляются автоматически, применяя функцию текущей даты (листы книги складского учета, журнала учета спецодежды, акты на списание). Для устранения ошибок также в экранных формах применяются поля со списками (фиксированными –

например, поле вид помещения в экранной форме «Список помещений», и на основе справочных данных – например, поле «Материальная ценность» в экранной форме «Список материальных ценностей к учету»).

Так же обеспечивается защита информации. Если раньше хранящиеся только на бумажном носителе данные могли быть утеряны или по прошествии времени (из-за внесения изменений, небрежного обращения) стать нечитабельными, то хранение информации в электронном виде само по себе устранит такого рода проблемы.
При внедрении проекта будет получен и социальный эффект (в данном случае для заместителя директора школы в виде повышения производительности труда), состоящий в том, что ответственное за материально-техническое обеспечение учебного процесса лицо, затратив меньшее количество времени на оформление документов, сможет более качественно выполнить свои обязанности. Высвободившееся время может быть перераспределено на выполнение других функций по должностным инструкциям работника.

Контрольные вопросы

1. Как определить стоимостные затраты на проект внедрения информационной системы (ИС)?
2. Какие факторы следует учесть при расчете стоимостей проекта внедрения ИС?
3. Как определить сроки окупаемости проекта внедрения ИС?
4. Какие методы расчета стоимости и сроков окупаемости проекта внедрения ИС существуют?
5. Каким образом можно учесть возможные риски и неопределенности при расчете стоимостей и сроков окупаемости проекта внедрения ИС?
6. Как влияют изменения в исходных данных, например, стоимость или объемы продаж, на расчеты стоимости и сроков окупаемости проекта внедрения ИС?
7. Как оценить эффективность проекта внедрения ИС на основе стоимостных показателей, таких как ROI (Return on Investment) или NPV (Net Present Value)?
8. Каким образом можно учесть дополнительные выгоды, такие как повышение эффективности работы или улучшение качества услуг, при расчете стоимостей и сроков окупаемости проекта внедрения ИС?
9. Какие инструменты и методики могут помочь в проведении стоимостной оценки проекта внедрения ИС и расчете сроков окупаемости?
10. Какие факторы следует учесть при прогнозировании будущих изменений стоимостей и сроков окупаемости внедрения ИС?

работа №27

ИЕ ДИАГРАММ ПОТОКОВ ДАННЫХ И ГЕНЕРАЦИЯ КОДА

Цель работы: получение навыков построения диаграмм потоков данных.

Порядок выполнения работы

−выполнить задание;
−показать преподавателю;
−ответить на вопросы преподавателя.
Время выполнения: 2 ч

Теоретическая часть

Понятие диаграммы потоков данных. Элементы диаграммы потоков данных. Хранилищаданных. Потокиуправления.

Рабочее задание

Задание № 1. Ознакомиться с методологией построения диаграмм потоков данных.
Диаграммы потоков данных (DataFlowDiagrams – DFD) используются для описания движения документов и обработки информации как дополнение к IDEF0. В отличие от IDEF0, где система рассматривается как взаимосвязанные работы, стрелки в DFD показывают лишь то, как объекты (включая данные) движутся от одной работы к другой. DFD отражает функциональные зависимости значений, вычисляемых в системе, включая входные значения, выходные значения и внутренние

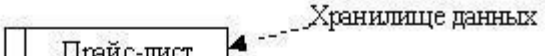
хранилища данных. DFD – это граф, на котором показано движение значений данных от их источников через преобразующие их процессы к их потребителям в других объектах.
DFD содержит процессы, которые преобразуют данные, потоки данных, которые переносят данные, активные объекты, которые производят и потребляют данные, и хранилища данных, которые пассивно хранят данные.
Диаграмма потоков данных содержит:

- процессы, которые преобразуютданные;
- потоки данных, переносящиеданные;
- активные объекты, которые производят и потребляютданные;
- хранилища данных, которые пассивно хранятданные.

Процесс DFD преобразует значения данных и изображается в виде эллипса, внутри которого помещается имя процесса (рисунок 11).
Рисунок 11

Поток данных соединяет выход объекта (или процесса) с входом другого объекта (или процесса) и представляет собой промежуточные данные вычислений. Поток данных изображается в виде стрелки между производителем и потребителем данных, помеченной именами соответствующих данных. Дуги могут разветвляться или сливаться, что означает соответственно разделение потока данных на части либо слияние объектов.
Активным объектом является объект, который обеспечивает движение данных, поставляя или потребляя их. Хранилище данных – это пассивный объект в составе DFD, в котором данные сохраняются для последующего доступа (рисунок 12).

Рисунок 12
Хранилища данных. Хранилище данных – это пассивный объект в составе DFD, в котором данные сохраняются для последующего доступа. Хранилище данных допускает доступ к хранимым в нем данным в порядке, отличном от того, в котором они были туда помещены. Агрегатные хранилища данных, как, например, списки и таблицы, обеспечивают доступ к данным в порядке их поступления, либо по ключам (рисунок 13).
Потоки управления. DFD показывает все пути вычисления значений, но не показывает, в каком порядке значения вычисляются. Решения о порядке вычислений связаны с управлением программой, которое отражается в динамической модели. Эти решения, вырабатываемые специальными функциями, или предикатами, определяют, будет ли выполнен тот или иной процесс, но при этом не передают процессу никаких данных, так что их включение вфункциональную модель необязательно. Тем не менее, иногда бывает полезно включать указанные предикаты в функциональную модель, чтобы в ней были отражены условия выполнения соответствующего процесса. Функция, принимающая решение о запуске процесса, будучи включенной в DFD, порождает в диаграмме поток управления и изображается пунктирной стрелкой (рисунок 14).
Рисунок13



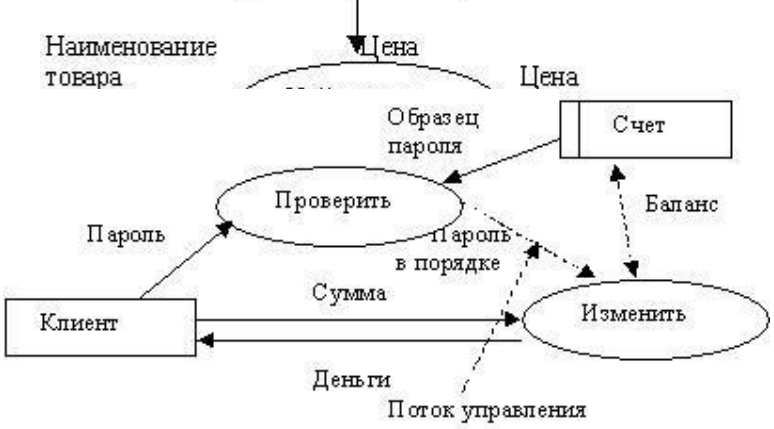


Рисунок14

Первым шагом при построении иерархии DFD является построение контекстных диаграмм. Обычно при проектировании относительно простых информационных систем строится единственная контекстная диаграмма со звездообразной топологией, в центре которой находится так называемый главный процесс, соединенный с приемниками и источниками информации, посредством которых с системой взаимодействуют пользователи и другие внешние системы.

Если же для сложной системы ограничиться единственной контекстной диаграммой, то она будет содержать слишком большое количество источников и приемников информации, которые трудно расположить на листе бумаги нормального формата, и, кроме того, главный единственный процесс не раскрывает структуры распределенной системы.

Для сложных информационных систем строится иерархия контекстных диаграмм. При этом контекстная диаграмма верхнего уровня содержит не главный единственный процесс, а набор подсистем, соединенных потоками данных. Контекстные диаграммы следующего уровня детализируют контекст и структуру подсистем.

При построении иерархии DFD переходить к детализации процессов следует только после определения содержания всех потоков и накопителей данных, которое описывается при помощи структур данных. Структуры данных конструируются из элементов данных и могут содержатьальтернативы, условные вхождения и итерации. Условное вхождение означает, что данный компонент может отсутствовать в структуре. Альтернатива означает, что в структуру может входить один из перечисленных элементов. Итерация означает вхождение любого числа элементов в указанном диапазоне. Для каждого элемента данных может указываться его тип (непрерывные или дискретные данные). Для непрерывных данных может указываться единица измерения (кг, см и т.п.), диапазон значений, точность представления и форма физического кодирования. Для дискретных данных может указываться таблица допустимыхзначений.

Задание № 2. Проанализируйте пример построения диаграммы потоков данных (рисунок 15).

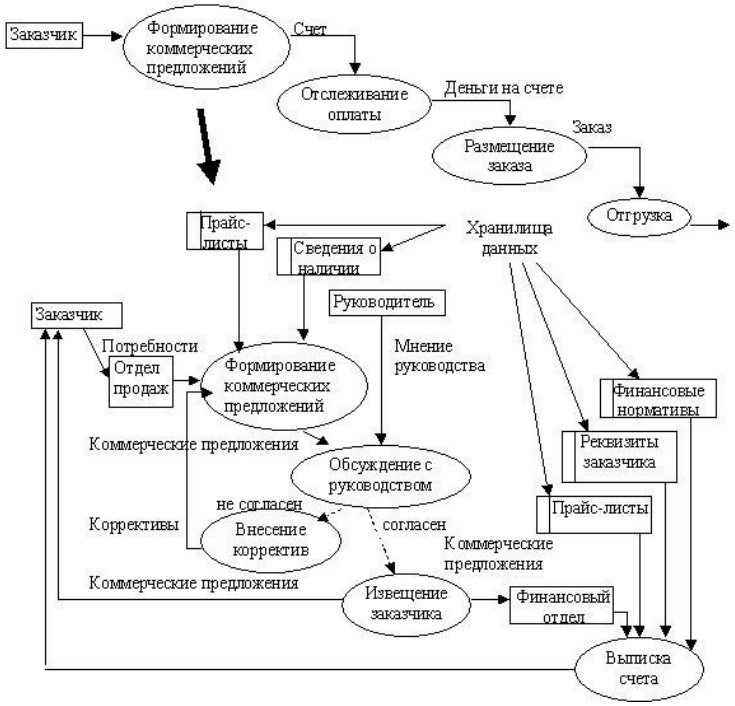


Рисунок 15

Задание № 3. Постройте диаграмму потоков данных для выбранной информационной системы (практическая работа №11).

Задание № 4. Оформите отчет.

Контрольные вопросы

ое диаграмма потоков данных (DFD) и какую роль она играет в анализе и проектировании систем?

сновные компоненты включает в себя диаграмма потоков данных?

бразом определяются и представляются потоки данных на диаграмме?

значаются процессы или функции на диаграмме потоков данных?

имволы используются для обозначения хранилищ данных на диаграмме?

зываются компоненты на диаграмме потоков данных, чтобы показать потоки данных между ними?

бразом можно представить разные уровни детализации на диаграмме потоков данных?

ользовать диаграммы потоков данных для выявления потенциальных проблем или улучшений в системе?

бразом диаграммы потоков данных могут быть использованы для документации и коммуникации с заинтересованными сторонами?

трументы или программное обеспечение могут помочь в построении диаграмм потоков данных?

работа №28

А И НАСТРОЙКА СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ С РАЗГРАНИЧЕНИЕМ РОЛЕЙ

Цель работы: получение навыков установки и настройки системы контроля версий.

Порядок выполнения работы

−выполнить задание;

−показать преподавателю;

−ответить на вопросы преподавателя.

Время выполнения: 2 ч

часть

Понятие системы контроля версий (СКВ), решаемые задачи.

Основные понятия СКВ и их отношения: хранилище, commit, история, рабочая копия. Отличия централизованных и децентрализованных СКВ. Примеры СКВ каждого вида. Действия с СКВ при единоличной работе с хранилищем.

Порядок работы с общим хранилищем в централизованной СКВ.

Рабочее задание

Задание № 1. Изучите систему контроля версий, установленную на компьютере (например, TortoiseSVN). При необходимости установите систему контроля версий TortoiseSVN. Опишите основные возможности системы контроля версий.

Задание № 2. Создайте новый проект. Создайте локальный репозиторий для своего проекта.

Удалите созданный проект на своем компьютере и обновите проект из репозитория.

Задание № 3. Внесите изменения в файлах с исходными кодами и сохраните изменения в репозитории. Обновите файлы с исходными кодами из репозитория. Внесите изменения в файлах с исходными кодами таким образом, чтобы у двух участников проекта изменения были в одном и том же файле. Попытайтесь сохранить изменения в репозитории. Устраните обнаруженные конфликты версий. Повторно сохраните изменения в репозитории. Создайте отдельную ветку проекта. Внесите изменения в файлы с исходнымикодами.

Задание № 4. Объедините созданную на предыдущем шаге ветку с основной веткой проекта. Выведите на экран данные изменений файла, в котором было наибольшее количество изменений. Отобразите на экране сравнение фала до и после внесения одного из изменений.

Задание № 5. Создайте репозитарий в сети Интернет. Удалите созданный проект на своем компьютере и обновите проект из репозитория. Внесите изменения в файлах с исходными кодами и сохраните изменения в репозитории. Обновите файлы с исходными кодами из репозитория. Внесите изменения в файлах с исходными кодами таким образом, чтобы у двух участников проектаизменениябыливодномитомжефайле.Попытайтесьсохранитьизмененияврепозитории.

Устраните обнаруженные конфликты версий. Повторно сохраните изменения в репозитории. Создайте отдельную ветку проекта. Внесите изменения в файлы с исходными кодами.

Задание № 6. Оформите отчет.

Контрольные вопросы

ое система контроля версий (Version Control System, VCS) и для чего она используется?

сновные виды систем контроля версий существуют и в чем их отличия?

ановить систему контроля версий на свой компьютер?

айлы и папки нужно исключить из контроля версий?

дать новый репозиторий в системе контроля версий?

троить основные параметры системы контроля версий, такие как имя пользователя и адрес электронной почты?

авить файлы в репозиторий и сделать первый коммит?

дать новую ветку (branch) в системе контроля версий и переключиться на нее?

отать с различными версиями файлов, отменять изменения и восстанавливать предыдущие версии?

дничать с другими разработчиками и обмениваться изменениями с помощью системы контроля версий?

Практическая работа №29

ОЕКТИРОВАНИЕ И РАЗРАБОТКА ИНТЕРФЕЙСАПОЛЬЗОВАТЕЛЯ

Цель работы: получение навыков проектирования и разработки интерфейса пользователя.

Порядок выполнения работы:

−выполнить задание;
−показать преподавателю;
−ответить на вопросы преподавателя.
Время выполнения: 2 ч

Теоретическая часть

Понятие пользовательского интерфейса. Виды пользовательских интерфейсов.
Основные элементы пользовательского интерфейса. Требования к разработке пользовательского интерфейса.

Рабочее задание

Задание № 1. Настроить среду разработки VisualStudio. Создать приложение для Windows, которое имитирует игровой автомат со «счастливыми» числами. Программа должна иметь следующий интерфейс (рисунок 16).
При нажатии на кнопку «Крутить» должны генерироваться три случайных числа от 0 до 9. Если хотя бы одно из них равно семи, на форме должны появляться надпись «Счастливая семерка» и картинка с изображением человека, платящего игроку деньги при выигрыше.Принажатии на кнопке «Выход» программа должна завершать работу. Решение сохранить под именем
«Игра». Создать исполняемый файл приложения.

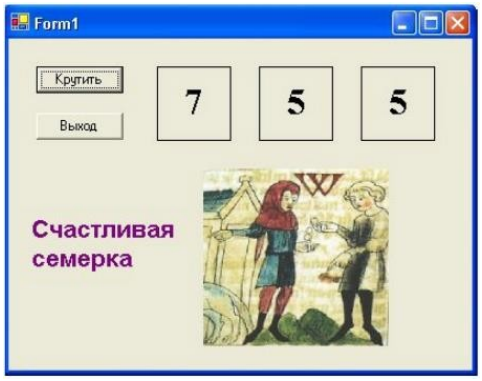


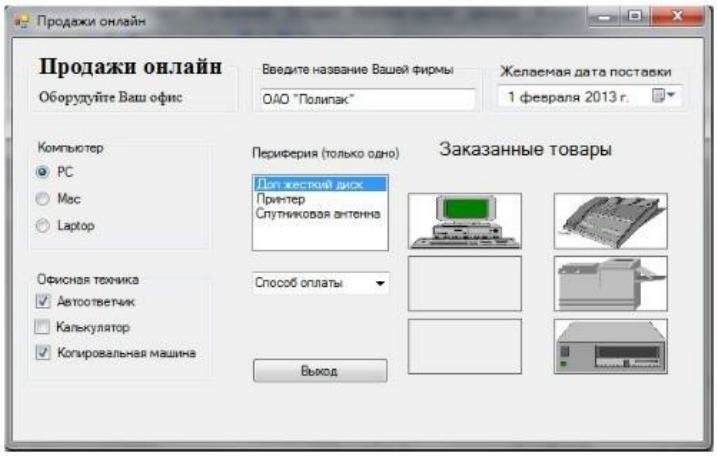
Рисунок 16

Задание № 2. Добавить в созданную форму метку и организовать отображение на ней процента выигрышей по отношению к общему числу нажатий на кнопку «Крутить».

Задание № 3. Добавить в программу оператор Randomize для того, чтобы программа при каждом запуске выдавала новую последовательность случайных чисел.

Задание № 4. Создать приложение для Windows «Продажи он-лайн», которое позволяет выбрать для заказа компьютер, офисную технику и периферийные устройства с выводом в форму изображения выбранного оборудования, указать способ оплаты и желаемую дату поставки. Возможные способы оплаты: рубли, доллары США, английские фунты. При выборе способа оплаты должно появляться его символическое изображение. Пользователь, выбрав товары для заказа, вводит название фирмы. Рекомендуемый интерфейс приложения приведен на рисунке 17.

Рисунок 17



Решение сохранить под именем «Продажи». Создать исполняемый файл приложения.

Задание № 5. Добавить в список офисной техники «МФУ» и добавить еще один объект PictureBox для отображения рисунка МФУ. Соответствующим образом изменить программный код.

Задание № 6. Добавить в способы оплаты «Чек».

Контрольные вопросы

акие основные принципы проектирования интерфейса пользователя следует учитывать при разработке приложений на C#?

ак создать пользовательский интерфейс в C# с использованием Windows Forms или WPF?

акие элементы управления (controls) доступны в C# для создания интерактивного интерфейса пользователя?

аким образом можно настроить внешний вид элементов управления и стилей в C#?

ак обрабатывать события (events) интерфейса пользователя в C#?

ак обеспечить валидацию ввода данных в интерфейсе пользователя на C#?

акой подход использовать для создания многопоточного интерфейса пользователя в C#?

ак интегрировать различные типы мультимедиа (например, изображения, звук, видео) в интерфейс пользователя на C#?

аким образом можно улучшить навигацию и поток работы пользователей в

интерфейсе C#?
организовать тестирование и отладку интерфейса пользователя в C# для обеспечения качества и удобства использования?

Практическая работа №30
РЕАЛИЗАЦИЯ АЛГОРИТМОВ ОБРАБОТКИ ЧИСЛОВЫХ ДАННЫХ. ОТЛАДКА ПРИЛОЖЕНИЯ

Цель работы: получение навыков реализации алгоритмов обработки числовых данных, отладки приложений.

Порядок выполнения работы:

−выполнить задание;
−показать преподавателю;
−ответить на вопросы преподавателя.

Время выполнения: 2 ч

Теоретическая часть

Элементы управления, используемые для обработки числовыхданных.

Рабочее задание

Задание № 1. Разработать приложение Windows, которое по заданным значениям: цены покупки, суммы первоначального платежа, годовой процентной ставки и срока кредита рассчитывает размер ежемесячных выплат по кредиту, а также строит схему платежей за каждый период (месяц) с разделением на основные платежи и платежи по процентам. Рассчитать также сумму всех основных платежей (для контроля) и сумму платежей по процентам (размер переплаты). Рекомендуемый интерфейс приложения показан на рисунке18.

Решение сохранить под именем «Платежи по кредиту».

Задание № 2. Внесите изменения в программный код так, чтобы в схеме платежей в 4-ом столбце отображалась общая сумма платежа за каждый период.

Задание № 3. Внесите изменения в форму и программный код так, чтобы платежи по кредиту осуществлялись не ежемесячно, а ежеквартально.

Задание № 4. Предусмотрите возможность пересмотра схемы платежей на оставшиеся периоды, если в некоторый период внесен платеж больше требуемой суммы. Рассмотреть такую схему погашения, при которой не уменьшается срок погашения кредита, а уменьшается сумма периодического платежа в последующих периодах.

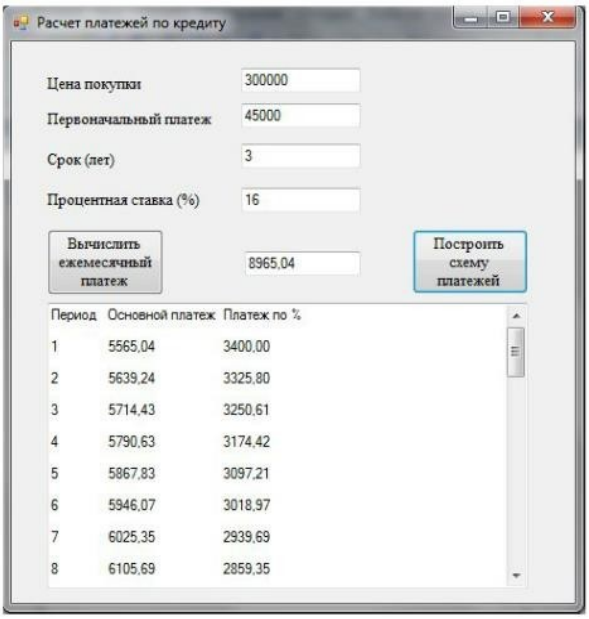


Рисунок 18

льные вопросы

акие методы и алгоритмы используются для сортировки числовых данных?

то такое поиск в массиве и какие методы и алгоритмы используются для его реализации?

акие алгоритмы применяются для поиска минимума и максимума в числовых данных?

акие алгоритмы и методы используются для обработки числовых данных в формате матриц или таблиц?

акие алгоритмы применяются для вычисления суммы или среднего значения числовых данных?

ак реализовать операции поиска, добавления, удаления и обновления элементов в динамической структуре данных, такой как список или дерево?

акие алгоритмы используются для решения задач аппроксимации и интерполяции числовых данных?

акие методы применяются для анализа и фильтрации временных рядов или сигналов?

ак реализовать алгоритмы поиска паттернов или совпадений в числовых данных?

использовать алгоритмы машинного обучения и искусственного интеллекта для анализа и обработки числовых данных?

Практическая работа №31
ЕАЛИЗАЦИЯ АЛГОРИТМОВ ПОИСКА. ОТЛАДКА ПРИЛОЖЕНИЯ

Цель работы: получение навыков реализации алгоритмов поиска данных, отладки приложений.

Порядок выполнения работы:

−выполнить задание;
−показать преподавателю;
−ответить на вопросы преподавателя.

Время выполнения: 2 ч

Теоретическая часть Алгоритмы поиска в тексте. Алгоритмы поиска в массивах.

Рабочее задание

Задание № 1. Написать программу «Результаты сессии», которая для выбранной из списка группы запрашивает ввод:− списка группы;− количества и названий предметов, покоторым
данная группа сдавала экзамены в последнюю сессию; − оценок студентов по предметам.
Программа должна также:

− отображать результаты сессии по даннойгруппе;
− вычислять качество знаний (процент студентов, успевающих на «хорошо» и«отлично»);
− вычислять процент успеваемости в группе (процент студентов, сдавшихсессию);
− определять количество студентов, успевающих на«отлично».

Вычисление качества знаний, процента успеваемости и количества отличников оформить в виде соответствующих процедур – функций. По итогам сессии должна быть рассчитана стипендия. Размеры минимальной и повышенной стипендии вводятся с клавиатуры. Минимальную стипендию получают студенты, сдавшие сессию на «хорошо» и«отлично».
В программе должны быть созданы 3 формы: главная форма «Результаты сессии и расчет стипендии», форма для отображения результатов сессии и форма «Размер стипендии» (рисунки 19, 20, 21, 22, 23, 24, 25, 26).

Рисунок 19

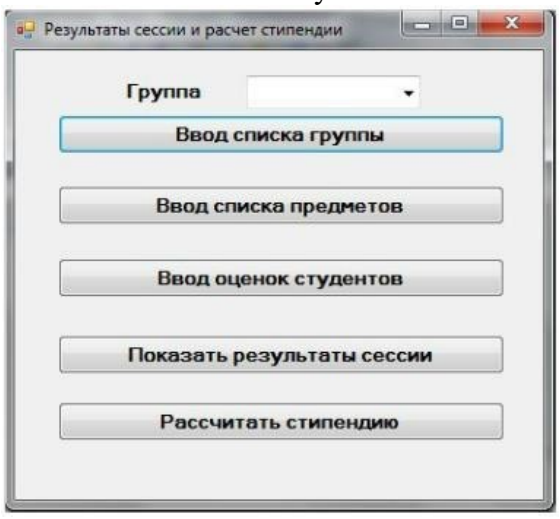


Рисунок 20

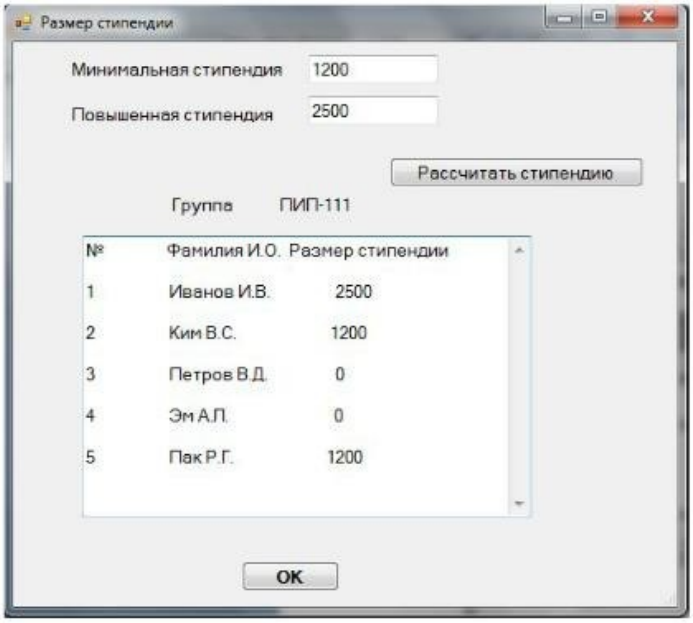


Рисунок21

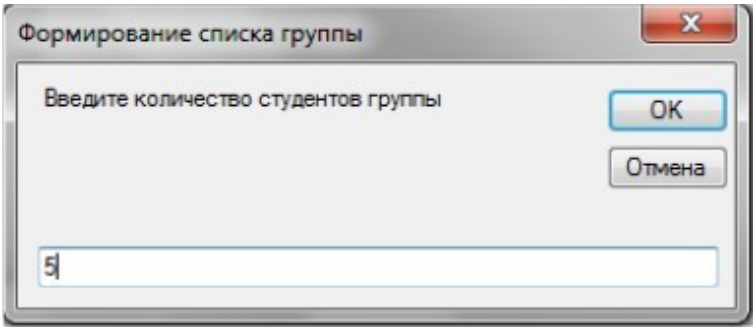


Рисунок22

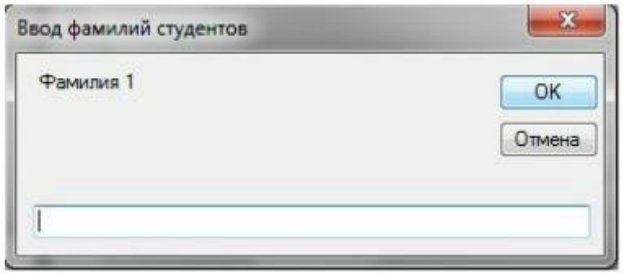


Рисунок2

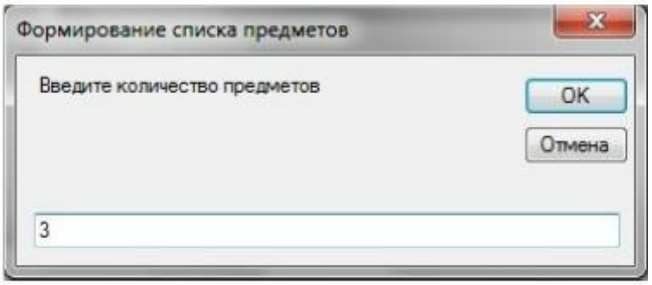


Рисунок 24

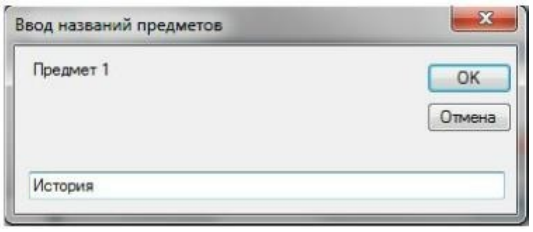


Рисунок25

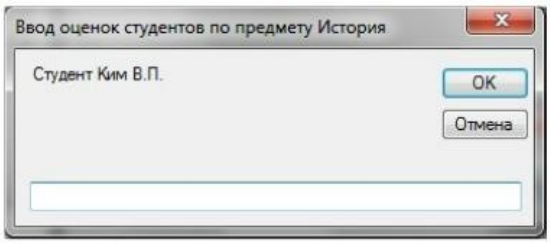


Рисунок26

Задание № 2. Написать программы, иллюстрирующие применение методов линейного поиска, поиска делением пополам, а также различные методы сортировки массивов.

льные вопросы

акие основные алгоритмы поиска используются для поиска данных в массиве или списке?

ак реализовать алгоритм бинарного поиска для быстрого поиска данных в отсортированном массиве?

акой алгоритм использовать для поиска подстроки или шаблона в строке или тексте?

акие алгоритмы могут быть использованы для поиска пути или оптимального пути в графе или сети?

ак реализовать алгоритм поиска наилучшего совпадения или наиболее подходящего элемента с использованием алгоритма наименьших квадратов (Least Squares)?

акие алгоритмы можно использовать для поиска определенной структуры данных, такой как дерево или граф, в другой структуре данных?

аким образом можно реализовать алгоритмы поиска с использованием хэш-функций и хэш-таблиц?

акие алгоритмы можно применять для анализа и классификации данных с использованием машинного обучения или интеллектуальных алгоритмов?

ак реализовать алгоритмы вероятностного поиска для поиска оптимальных решений или перебора значений в большом пространстве поиска?

ие алгоритмы можно использовать для поиска и фильтрации данных в режиме реального времени, например, в потоковых данных или онлайн-

системах?

Практическая работа №32
РЕАЛИЗАЦИЯ ОБРАБОТКИ ТАБЛИЧНЫХ ДАННЫХ. ОТЛАДКА ПРИЛОЖЕНИЯ

Цель работы: получение навыков обработки табличных данных, отладки приложений.

Порядок выполнения работы:

−выполнить задание;
−показать преподавателю;
−ответить на вопросы преподавателя.

Время выполнения: 2 ч

Теоретическая часть

Обработка табличных данных в приложениях.

Рабочее задание

Задание № 1. Организовать работу с базой данных Студенты, которая храниться в текстовом файле. При выборе в списке ComboBox определенной группы на форме Списки групп отобразит в сетке данных DataGridView только фамилии студентов данной группы. Рекомендуемый интерфейс приложения изображен на рисунке 27.

Задание № 2. Создать запрос, который будет отбирать из базы данных Студенты фамилии студентов заданного курса, записывать их вместе с названием группы во временный файл СтудентыВрем и отображать на форме с помощью элемента DataGridView.

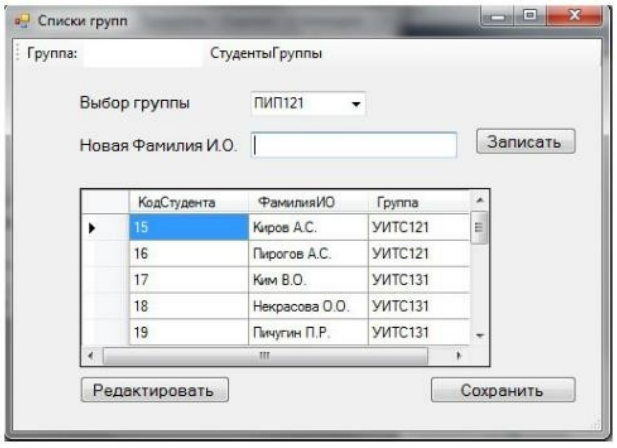


Рисунок 27

Контрольные вопросы

ак создать и инициализировать таблицу данных в C#?
ак добавить новые строки и столбцы в таблицу данных в C#?
аким образом можно обращаться к данным в таблице и изменять их значения в

C#?

ак отфильтровать и сортировать данные в таблице данных в C#?

ак реализовать поиск и выборку данных по определенным критериям или условиям в таблице данных в C#?

аким образом можно объединять, разделять или преобразовывать таблицы данных в C#?

ак выполнять агрегацию и вычисление статистических значений (например, сумма, среднее, максимум) в таблице данных в C#?

ак реализовать связь и отношения между таблицами данных в C#?

аким образом можно экспортировать и импортировать таблицы данных из различных форматов (например, CSV, Excel) в C#?

обрабатывать ошибки и обеспечить контроль целостности данных при работе с таблицами данных в C#?

Практическая работа №33
АЗРАБОТКА И ОТЛАДКА ГЕНЕРАТОРА СЛУЧАЙНЫХ СИМВОЛОВ

Цель работы: получение навыков разработки и отладки генератора случайных символов.

Порядок выполнения работы

−выполнить задание;
−показать преподавателю;
−ответить на вопросы преподавателя.

Время выполнения: 2 ч

Теоретическая часть

Понятие генератора случайных символов. Управление генератором случайных символов.

Рабочее задание

Задание № 1. Разработать генератор случайных чисел.

Случайные числа в языке программирования C++ могут быть сгенерированы функцией rand() из стандартной библиотеки C++. Функцияrand() генерируетчисла в диапазоне от 0 до
RAND_MAX. RAND_MAX – это константа, определённая в библиотеке <cstdlib>. Для MVS RAND_MAX = 32767, но оно может быть и больше, в зависимости от компилятора. Ниже показана простая программка, использующая генератор случайных чиселrand():

```
#include "stdafx.h" #include<iostream>usingnamespacestd;
int main(intargc, char* argv[])
{
cout<< "RAND_MAX = " << RAND_MAX <<endl; // константа, хранящая максимальный предел из интервала случайных чисел
```

```
cout<< "random number = " << rand() <<endl; // запускгенератораслучайныхчисел
system("pause");
return 0;
}
```

Максимальное случайное число в примере – это 32767. Зачастую, нам не нужен такой большой диапазон чисел от 0 до RAND_MAX. Например, в игре «Наперстки» необходимо отгадать, под каким из трёх напёрстков спрятан шарик, то есть генерация чисел должна выполняться в пределе от 1 до 3-х. Бросая монету, может возникнуть только два случая, когда монета упадёт «орлом» или «решкой» вверх, нужный интервал – от 1 до 2. Возникает потребность в масштабировании интервала генерации случайных чисел. Для того чтобы масштабировать интервал генерации чисел нужно воспользоваться, операцией нахождения остатка от деления«%»:

```
// пример масштабирования диапазона генерации случайных чисел rand() % 3 +1 //
диапазон равен от 1 до 3 включительно
```

Число 3 является масштабируемым коэффициентом. То есть, какое бы не выдал число генератор случайных чисел rand() запись rand() % 3 в итоге выдаст число из диапазона от 0 до 2. Для того чтобы сместить диапазон, мы прибавляем единицу, тогда диапазон изменится на такой – от 1 до 3 включительно.

Задание № 2. Разработать программу, использующую масштабируемый генератор случайных чисел. Ниже показан код программы, которая несколько раз запускает функцию rand().

```
// rand_ost.cpp: определяет точку входа для консольного приложения.
#include "stdafx.h" #include <iostream> using namespace std;
int main(intargc, char* argv[])
{
cout<< "1-random number = " << 1 + rand() % 3 <<endl; // первый запуск генератора
случайных чисел
cout<< "2-random number = " << 1 + rand() % 3 <<endl; // второй запуск генератора
случайных чисел
cout<< "3-random number = " << 1 + rand() % 3 <<endl; // третий запуск генератора
случайных чисел
cout<< "4-random number = " << 1 + rand() % 3 <<endl; // четвёртый запуск генератора случайных
чисел
cout<< "5-random number = " << 1 + rand() % 3 <<endl; // пятый запуск генератора случайных чисел
cout<< "6-random number = " << 1 + rand() % 3 <<endl; // шестой запуск генератора случайных
чисел
cout<< "7-random number = " << 1 + rand() % 3 <<endl; // седьмой запуск генератора случайных
чисел
cout<< "8-random number = " << 1 + rand() % 3 <<endl; // восьмой запуск генератора случайных
чисел
system("pause"); return 0;
}
```

При повторном запуске программы, печатаются те же самые числа. Суть в том, что функция rand() один раз генерирует случайные числа, а при последующих запусках программы всего лишь отображает сгенерированные первый раз числа. Такая особенность функции rand() нужна для того, чтобы можно было правильно отладить разрабатываемую программу. При отладке программы, внеся какие-то изменения, необходимо удостовериться, что программа срабатывает правильно, а это возможно, если входные данные остались те же, то есть сгенерированные числа. Когда программа успешно отлажена, нужно, чтобы при каждом выполнении программы генерировались случайные числа. Для этого нужно воспользоваться функцией srand() из стандартной библиотеки C++. Функция srand() получив целый положительный аргумент типа unsigned или unsignedint

(без знаковое целое) выполняет рандомизацию, таким образом, чтобы при каждом запуске программы функция srand() генерировала случайные числа. Программа, использующая функцию srand() для рандомизации генератора случайных чисел rand():

```
// srand.cpp: определяет точку входа для консольного приложения.
#include "stdafx.h" #include <iostream> using namespace std;
int main(intargc, char* argv[])
{
unsigned rand_value = 11;
srand(rand_value); // рандомизациягенератораслучайныхчиселcout<< "rand_value = "
<<rand_value<<endl;
cout<< "1-random number = " << 1 + rand() % 10 <<endl; // первый запускгенератора
случайных чисел
cout<< "2-random number = " << 1 + rand() % 10 <<endl; // второй запуск генератора
случайных чисел
system("pause"); return 0;
}
```

Задание № 3. Разработать обобщённый пример использования автоматического генератора случайных чисел с масштабированием. Пример работы программы:

```
// srand_time.cpp: определяет точку входа для консольного приложения. #include "stdafx.h"
#include <iostream>
#include <ctime> using namespace std;
int main(intargc, char* argv[])
{
srand( time( 0 ) ); // автоматическаярандомизациясout<< "rand_value = " << 1 + rand() % 10
<<endl; system("pause");
return 0;
}
```

Теперь при каждом срабатывании программы будут генерироваться совершенно случайные числа в интервале от 1 до 10, включительно.

Задание № 4. Разработать генератор случайных символов. Сформировать случайную символьную последовательность.

Контрольные вопросы

ак создать генератор случайных символов в выбранном языке программирования?

ак сгенерировать случайный символ из определенного набора символов?

ак убедиться, что сгенерированный символ является уникальным или не повторяется в предыдущих генерациях?

ак создать генератор случайной строки из символов определенной длины?

ак обеспечить настройку вероятности появления определенных символов при генерации случайных символов?

аким образом можно отлаживать генератор случайных символов и проверять правильность его работы?

ак создать функцию для генерации случайной последовательности символов заданной длины?

аким образом можно обрабатывать возможные ошибки и исключения при генерации случайных символов?

ак проверить равномерность распределения случайных символов, чтобы

избежать предвзятости или смещения в результате генерации?
реализовать генератор случайных символов с заданными параметрами, такими как длина строки, допустимые символы и особые условия (например, уникальность символов)?

Практическая работа №34
ИНТЕГРАЦИЯ МОДУЛЯ В ИНФОРМАЦИОННУЮ СИСТЕМУ

Цель работы: получение навыков интеграции модулей в информационную систему.

Порядок выполнения работы

−выполнить задание;
−показать преподавателю;
−ответить на вопросы преподавателя.

Время выполнения: 2 ч

Теоретическая часть

Понятие модуля.
Управление модулями.

Рабочее задание

Задание № 1. Создать файл, содержащий сведения о сдаче студентами сессии. Структура записи: индекс группы, фамилия студента с его инициалами, оценки по четырем экзаменам и пяти зачетам («з» означает зачет, «н» – незачет). Экзамены и зачеты нумеровать цифрами. Количество записей в файле не менее двадцати.

При разработке приложения использовать стандартные модули.

Задание № 2. Разработать программу, интегрирующую модули из приложения, разработанного в рамках задания №1, выводящую следующуюинформацию:

– фамилии неуспевающих студентов с указанием индексов групп и видазадолженности;
– фамилии студентов, сдавших все зачеты и получившие на экзаменах четверки ипятерки;
– средний бал, полученный каждымстудентом.

льные вопросы

аким образом можно интегрировать модули в существующую информационную систему?
акие методы или протоколы можно использовать для связи и обмена данными между модулями в информационной системе?
ак реализовать механизм автоматического обнаружения и регистрации модулей при интеграции в информационную систему?
аким образом можно обеспечить безопасность и аутентификацию при обмене данными между модулями в информационной системе?
ак реализовать механизм публикации и подписки на события между модулями в

информационной системе?
ак управлять зависимостями и версиями модулей при их интеграции в информационную систему?
аким образом можно мониторить и отслеживать работу интегрированных модулей в информационной системе?
ак реализовать механизм конфигурации и настройки параметров интегрированных модулей в информационной системе?
ак обеспечить масштабируемость и гибкость в интеграции модулей в информационную систему для возможности добавления новых модулей в будущем?
оценить и улучшить производительность и эффективность интеграции модулей в информационной системе?

Практическая работа №35
ЗРАБОТКА ПРИЛОЖЕНИЙ ДЛЯ МОДЕЛИРОВАНИЯ ПРОЦЕССОВ И ЯВЛЕНИЙ. ОТЛАДКА ПРИЛОЖЕНИЯ

Цель работы: получение навыков разработки и отладки приложений для моделирования процессов и явлений.

Порядок выполнения работы:

−выполнить задание;
−показать преподавателю;
−ответить на вопросы преподавателя.

Время выполнения: 2 ч

Теоретическая часть

Понятие модели.
Моделирование процессов и явлений.
Технологии моделирования процессов и явлений в приложениях.

Рабочее задание

Задание № 1. Разработать физико-математическую модель системы при сободном падении физического тела, брошенного с высоты h и падающего свободно в течение t времени. При построении модели принять следующие гипотезы:

1) падение происходит в вакууме (то есть коэффициент сопротивления воздуха равеннулю);
2) ветранет;
3) масса теланеизменна;
4) тело движется с одинаковым постоянным ускорением g в любойточке.

Слово "модель" (лат. modelium) означает "мера", "способ", "сходство с какой-то вещью".
Проблема моделирования состоит из трех взаимосвязанных задач: построение новой (адаптация известной) модели; исследование модели (разработка метода исследования или адаптация, применение известного); использование (на практике или теоретически) модели.

Схема построения модели M системы S с входными сигналами X и выходными сигналами Y изображена на рисунке 28.

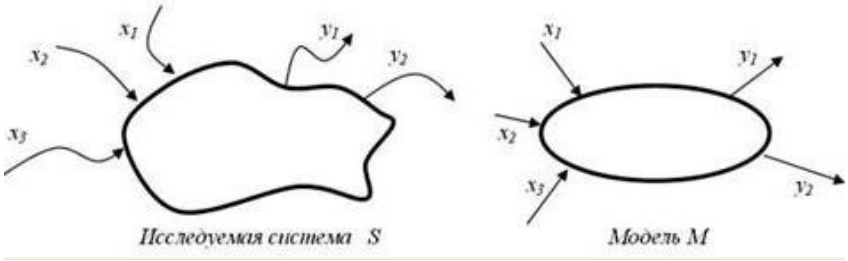


Рисунок 28

Если на вход M поступают сигналы из X и на входе появляются сигналы из Y, то задан закон, правило f функционирования модели, системы.

Классификацию моделей проводят по различным критериям.

Модель – статическая, если среди параметров описания модели нет (явно) временного параметра.

Модель – динамическая, если среди параметров модели явно выделен временной параметр.

Модель – дискретная, если описывает поведение оригинала лишь дискретно, например, в дискретные моменты времени (для динамической модели).

Модель – непрерывная, если описывает поведение оригинала на всем промежутке времени.

Модель – детерминированная, если для каждой допустимой совокупности входных параметров она позволяет определять однозначно набор выходных параметров; в противном случае – модель недетерминированная, стохастическая (вероятностная).

Модель – функциональная, если представима системой функциональных соотношений (например, уравнений).

Модель – теоретико-множественная, если представима некоторыми множествами и отношениями их и ихэлементов.

Модель – логическая, если представима предикатами, логическими функциями и отношениями.

Модель – информационно-логическая, если она представима информацией о составных элементах, подмоделях, а также логическими отношениями между ними.

Модель – игровая, если она описывает, реализует некоторую игровую ситуацию между элементами (объектами и субъектами игры).

Модель – алгоритмическая, если она описана некоторым алгоритмом или комплексом алгоритмов, определяющим ее функционирование, развитие. Введение такого, на первый взгляд, непривычного типа моделей (действительно, кажется, что любая модель может быть представлена алгоритмом ее исследования), на наш взгляд, вполне обосновано, так как не все модели могут быть исследованы или реализованыалгоритмически.

Модель – графовая, если она представима графом (отношениями вершин и соединяющих их ребер) или графами и отношениями между ними.

Модель – иерархическая (древовидная), если она представима иерахической структурой (деревом).

Модель – языковая, лингвистическая, если она представлена некоторым лингвистическим объектом, формализованной языковой системой или структурой. Иногда такие модели называют вербальными, синтаксическими и т.п.

Модель – визуальная, если она позволяет визуализировать отношения и связи моделируемой системы, особенно в динамике.

Модель – натурная, если она есть материальная копия оригинала.

Модель – геометрическая, если она представима геометрическими образами и отношениями между ними.

Модель – имитационная, если она построена для испытания или изучения, проигрывания возможных путей развития и поведения объекта путем варьирования некоторых или всех параметров модели.

Задание № 2. Разработать статическая модель движения тела по наклонной плоскости F = am. Динамическая модель типа закона Ньютона: F(t) = a(t)m(t) или, еще более точно и лучше, F(t)=s"(t)m(t). Если рассматривать только t = 0.1, 0.2, …, 1 (c), то модель St = gt2/2 или числовая последовательность S0 = 0, S1 = 0.01g/2, S2 = 0.04g, …, S10 = g/2 может служить дискретной

моделью движения свободно падающего тела. Модель S = gt2/2, 0 < t < 10 непрерывна на *промежутке времени (0;10).*

Задание № 3. Разработать модель популяции рыб, из которой в текущий момент времени изымается некоторое количество особей (идет лов рыбы). Динамика такой системы определяется моделью вида: xi + 1 = xi + axi – kxi, x0 = c , где k – коэффициент вылова (скорость изъятия особей). Стоимость одной пойманной рыбы равна b руб. Цель моделирования – прогноз прибыли при заданной квоте вылова. Для этой модели можно проводить имитационные вычислительные эксперименты и далее модифицировать модель, например следующим образом.

Эксперимент 1. Для заданных параметров a, c изменяя параметр k, определить его наибольшее значение, при котором популяция не вымирает.

Эксперимент 2. Для заданных параметров c, k изменяя параметр a, определить его наибольшее значение, при котором популяция вымирает.

Модификация 1. Учитываем естественную гибель популяции (за счет нехватки пищи, например) с коэффициентом смертности, равным, b: xi + 1 = xi + axi – (k + b)xi, x0 = c .

Модификация 2. Учитываем зависимость коэффициента k o от x (например, k = dx): $x_{i+1} = x_i + ax_i - dx_i^2, x_0 = c$.

льные вопросы

ак выбрать подходящую технологию для разработки приложения моделирования процессов и явлений?

аким образом можно создать модель процесса или явления в приложении для моделирования?

ак реализовать визуализацию моделирования процессов и явлений в приложении?

аким образом можно симулировать временные параметры и динамику в приложении моделирования?

ак обработать входные данные и параметры для моделирования процессов и явлений в приложении?

аким образом можно отлаживать и проверять правильность работы моделирования в приложении?

ак реализовать возможность настройки и изменения параметров моделирования в приложении?

аким образом можно обрабатывать и представлять результаты моделирования процессов и явлений в приложении?

ак оценить эффективность и точность моделирования в приложении и провести его валидацию?

реализовать возможность взаимодействия и интеграции с другими системами и сервисами в приложении моделирования процессов и явлений?

Практическая работа №36

ГРАММИРОВАНИЕ ОБМЕНА СООБЩЕНИЯМИ МЕЖДУ МОДУЛЯМИ

Цель работы: получение навыков программирования обмена сообщениями между модулями.

Порядок выполнения работы:

−выполнить задание;
−показать преподавателю;
−ответить на вопросы преподавателя.

Время выполнения: 2 ч

Теоретическая часть

Понятие и структура сообщения. Обмен сообщениями между модулями.

Рабочее задание

Задание № 1. Составить программу, помогающую сотрудникам Государственной инспекции безопасности дорожного движения (ГИБДД) обработать следующие данные: регистрационный номер автомобиля, марка автомобиля, цвет автомобиля, год выпуска, адрес владельца. Программа должна по требованию пользователя выдавать следующие сведения:

- адреса владельцев автомобилей заданной марки, определенногоцвета;
- все данные об автомобиле с заданным регистрационнымномером;
- *все данные об автомобилях с известной цифровой частью регистрационногономера.*

Задание № 2. Программу, разработанную в задании №1, разбить на модули. Например, создать такие модули, как главный (содержащий функцию main()), чтения из файла в массив структур, вывод на экран содержимого массива структур, сортировка данных (при необходимости), меню, формирование документов ит.д.

Задание № 3. Разработать схему межмодульных вызовов.

Задание № 4. Проанализировать способы передачи аргументов между функциями и целесообразность использования глобальных данных

Контрольные вопросы

1. Каким образом можно реализовать обмен сообщениями между модулями при помощи технологии программирования?
2. Как выбрать подходящий протокол обмена сообщениями для взаимодействия между модулями?
3. Как реализовать механизм отправки и получения сообщений между модулями в программе?
4. Как обеспечить синхронизацию и параллельную обработку сообщений между модулями?
5. Как управлять очередью сообщений и приоритетами обработки взаимодействия между модулями?
6. Каким образом можно реализовать сериализацию и десериализацию сообщений для передачи данных между модулями?
7. Как обработать ошибки и исключения при обмене сообщениями между модулями?
8. Как реализовать механизм подписки и рассылки сообщений между модулями для возможности реакции на определенные события?
9. Каким образом можно обеспечить безопасность и аутентификацию при обмене сообщениями между модулями?
10. Как оценить производительность и эффективность обмена сообщениями между модулями в программе и провести его оптимизацию?

Практическая работа №37

ОРГАНИЗАЦИЯ ФАЙЛОВОГО ВВОДА-ВЫВОДА ДАННЫХ

Цель работы: получить практические навыки программирования задач ввода-вывода с использованием файлов.

Порядок выполнения работы:

−выполнить задание;
−показать преподавателю;
−ответить на вопросы преподавателя.

Время выполнения: 2 ч

Теоретическая часть

Организация ввода и вывода. Файловая система

Операции ввода/вывода в языке С осуществляются через потоки. *Поток* - это логическое устройство, выдающее и принимающее информацию.

С потоком связано понятие внутреннего указателя, который определяет позицию, с которой начинается следующая операция чтения или записи. При каждой операции чтения или записи происходит автоматическое перемещение указателя.

В языке С (С++) формат стандартных файлов ввода/вывода описан в заголовочном файле stdio.h. Имена стандартных файлов ввода/вывода для языка С (С++) представлены в табл. 7.1. В момент начала выполнения программы на языке С (С++) автоматически открываются три потока: stdin, stdout, stderr.

да 7.1

Потоки, определяемые в языке С и С++

Имя стандартного файла	Описание
stdaux	Последовательный ввод/вывод
stderr	Выходной поток ошибок
stdin	Стандартный ввод
stdout	Стандартный вывод
stdprn	Вывод на принтер

С++ поддерживает всю систему ввода/вывода С и добавляет к ней дополнительные возможности, связанные в основном с вводом/выводом объектов. Описание средств для создания потоков в С++ представлено в заголовочном файле iostream.h. Когда начинает работать программа на С++, открываются потоки, приведенные в табл. 7.2.

Таблица 7.2

Потоки, определяемые в языке С++

Имя стандартного файла	Описание
cin	Стандартный ввод - клавиатура
cout	Стандартный вывод - экран
cerr	Стандартная ошибка - экран
clog	Буферизованная версия cerr - экран

Файловая система языков С и С++ состоит как бы из двух уровней: логических файлов и физических файлов, с которыми логические файлы всегда связаны.

Логический файл описывается как указатель на открываемый поток FILE * и служит средством взаимодействия с физическим файлом. Имя физического файла появляется в программе всего один раз, в тот момент, когда происходит открытие файла, осуществляемое функцией fopen() и одновременно его связывание с логическим файлом.

Основными действиями, производимыми над файлами, являются их открытие, обработка и закрытие. Обработка файлов может заключаться в считывании блока данных из потока в

оперативную память, запись блока данных из оперативной памяти в поток, считывание определенной записи данных из потока, занесение определенной записи данных в поток. При этом необходимо помнить, что понятие файла в памяти ЭВМ не определено, и приобретает смысл только после его связи с внешним физическим файлом.

Текстовые файлы

Тип FILE определяется в заголовочном файле stdio.h и обычно представляет собой структуру, содержащую параметры реализации потока, такие как адреса буферов, указатели позиций потока, маркеры ошибок потока и т.д.

При работе с дисковыми файлами в момент их открытия следует задать режим доступа, чтобы определить, к какому файлу осуществляется доступ: к текстовому или двоичному, а также способ доступа: чтение или запись. Все это выполняется функцией fopen(), имеющей синтаксис:

fopen("**имя_файла**", "**режим_доступа**")

Режимы доступа к файлам для функции fopen() приведены в табл. 7.3.

Таблица 7.3

Режимы доступа к файлам

Режим	Описание
r	Открыть файл только для чтения, модификации файла запрещены.
w	Создать новый файл только для записи. При попытке открыть таким способом существующий файл происходит перезапись файла. Чтение данных из файла запрещено.
a	Открыть файл для дозаписи. Если файла с указанным именем не существует, он будет создан.
r+	Открыть существующий файл для чтения и записи.
w+	Создать новый файл для чтения и записи.
a+	Открыть существующий файл для дозаписи и чтения.

Таким образом, чтобы открыть текстовый файл, например, для чтения, нужно произвести следующие действия:

```
FILE *ft;                  // объявили указатель на файловый поток
ft = fopen("inp_f.txt","r");     // открыли файл inp_f.txt
```

При попытке открыть существующий файл можно допустить ошибку в его имени или в указании пути к нужному файлу. Это вызывает ошибку исполнения программы. Следует предвидеть подобные ситуации и проводить проверку возможности открытия файла. Такую проверку осуществить довольно легко, так как функция fopen() возвращает значение указателя в случае успешного его открытия и значение NULL, если доступ к файлу невозможен. Поэтому достаточно написать:

```
if (ft = fopen("inp_f.txt","r") != NULL)
{  // обработка файла
   }
```

Текстовой файл состоит из последовательности символов, разбитой на строки путем использования управляющего символа \n. На диске текстовые файлы хранятся в виде сплошной последовательности символов, и их деление на строки становится заметным лишь в момент вывода на экран или печать, так как именно при выводе управляющие символы начинают выполнять свои функции. Текстовые файлы легко переносятся с одного типа компьютера на другой лишь в случаях, когда они содержат только символы, принадлежащие стандартному набору символов.

При работе с текстовыми файлами возможна их посимвольная или построчная обработка.

Основные методы обработки текстовых файлов

Файловые функции ввода/вывода fprintf() и fscanf() работают аналогично функциям printf() и scanf(), но имеют дополнительный аргумент, являющийся указателем на файловый поток.

Пример 7.1.Чтение одного элемента из файла, обработка и запись результата в текстовый файл.

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
void main()
{  FILE *f;
int dig;
   if (f = (fopen("inp_f","r")) == NULL)      // открытьфайлдлячтения
{ printf("Невозможно открыть файл!\n");
exit(0);
        }
fscanf(f, "%d", &dig);                    // считать значение dig из файла
fclose(f);                          // закрыть файл
f = fopen("out_f","w");                          // открыть файл для записи
fprintf(f, "Мы прочитали число %d", dig);
fclose(f);                          // закрыть файл
}
```

Мы использовали один и тот же указатель на файловый поток дважды, так как прежде, чем обращаться к нему вторично, закрыли связанный с ним файл и освободили, таким образом, указатель.

В приведенном примере имя файла было записано непосредственно в операторе открытия файла. Но можно сообщить имя открываемого файла, введя его с клавиатуры или пользуясь аргументами командной строки.

Пример 7.2.Написать программу, которая сжимает содержимое файла, записывая в выходной файл лишь каждый третий символ из входного файла.

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <string.h>
void main(intargc, char *argv[])
{ FILE *f_in, *f_out;
intch;
  char *name;                   // имявходногофайла
int count = 0;                  // счетчикэлементов
  if (argc<2)                   // в командной строке нет имени
{  printf("Введите имя входного файла");
gets(name);
       }
elsename = argv[1];             // взять имя из командной строки
if ((f_in = fopen(name, "r")) != NULL)
{ strcat(name, ".out");                 // добавляет расширение .out
                // к имени файла
f_out = fopen(name, "w");    // открывает файл для записи
while((ch = fgetc(f_in)) != EOF)
if (count++ % 3 == 0)
fputc(ch, f_out);        // выводит каждый третий символ
fclose(f_in);
fclose(f_out);
```

```
}
elseprintf("Невозможно открыть файл\n ");
}
```

При работе с текстовыми файлами возможна не только поэлементная обработка файлов, но и построчная.

Пример 7.3.Построчное чтение информации из входного файла и вывод ее на экран как на стандартное устройство вывода.

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <string.h>
void main(intargc, char *argv[])
{ FILE *f_in;
charbuffer[256];        // максимальная длина строки - 255 символов
char *name;    // имя входного файла
if (argc<2)                // в командной строке нет имени
{  printf("Введите имя входного файла");
gets(name);
        }
elsename = argv[1];              // взять имя из командной строки
if ((f_in = fopen(name, "r")) != NULL)
{ while (fgets(buffer,255,f_in)) != NULL)
{ fputs(buffer, stdout);
fputc('\n', stdout);
                }
fclose(f_in);
         }
     else printf("Невозможнооткрытьфайл\n ");
}
```

В цикле while присутствуют две файловые функции работы со строками: fgets() для чтения строки символов в буфер и fputs() - для записи содержимого буфера в файл.

Закрыть файл не менее важно, чем открыть его, так как в этот момент происходит заполнение ячейки таблицы размещения файлов значением, которое является признаком завершения файла и установка атрибутов файла.

Кроме того, вывод текстового файла буферизован. Это значит, что в тот момент, когда работает оператор записи в файл, фактическая запись может и не происходить, поскольку реально сначала происходит заполнение данными текстового буфера, а потом его содержимое записывается на диск. Запись буфера происходит как только он окажется полностью заполненным или при выполнении специальных команд принудительной записи на диск. Процесс записи недозаполненного буфера на диск называется *флэшированием* и обычно выполняется с помощью функции fflush(f_out). При необходимости завершить работу сразу со всеми открытыми файлами пользуются функцией flushall().

Закрытие файла посредством функции fclose(f_out) также включает процесс флэширования, то есть перенос информации из буфера на диск.

Доступ к элементам текстовых файлов возможен только в последовательном режиме как при записи файла, так и при его чтении.

задание

Напишите программу согласно Вашему варианту задания.

Варианты заданий

Номер варианта	Задание

1, 14	Случайным образом создать таблицу пар целочисленных значений и записать её в текстовый файл в виде: XY 5 1 2 8 12 3 - - - - Считать из файла пары значений и в тех из них, где X>Y , поменять значения X и Y местами. Результат записать в другой текстовый файл такого же формата.
2, 15	Ввести с клавиатуры попарно значения вещественного типа и записать их в текстовый файл в виде таблицы следующего формата: X :Y 2.1 :3.7 6.2 : 5.4 --- - --- Считать из файла полученные пары значений и создать из них другой файл вида: sin(x) :cos(y) значение sin(2.1) : значение cos(3.7) ---------- - ----------
Номер варианта	Задание
3, 16	Случайным образом создать таблицу пар значений и записать её в текстовый файл в виде: n * c 5 * m 7 * a 3 * q --- Считать из файла пары значений и создать из них другой текстовый файл вида mmmmm aaaaaaa qqq
4, 17	Случайным образом создать таблицу пар целочисленных значений и записать её в текстовый файл в виде: XY 51 2 8 12 3 - - - - Считать из файла пары значений и в тех из них, где X кратен Y , пометить строку таблицы: XY 51 *** 2 8 12 3 *** - - в том же файле.
5, 18	Случайным образом создать таблицу пар значений и

	записать её в текстовый файл в виде: abc 5.2 4.6 2.5 можно 1.2 8.9 2.3 --- Считать из файла записанные данные и определить, можно ли построить треугольник с такими сторонами. Пометить соответствующие строки таблицы (в том же файле).
Номер варианта	Задание
6, 19	Создать текстовый файл, содержащий целочисленные значения, следующего формата 5 21 4 37 52 9 . . .Определить, являются ли значения, находящиеся в файле, упорядоченными по возрастанию.
7, 20	Создать текстовый файл, содержащий вещественные значения, следующего формата 5.3 21.4 37.4 52.6 9.2 . . . Считать из файла записанные данные и определить максимальное значение. Если оно находится в первой половине файла, заменить его суммой последующих элементов, если во второй – суммой предыдущих элементов.
8, 21	Случайным образом создать таблицу пар целочисленных значений и записать её в текстовый файл в виде: XY 5 25 1 3 49 7 - - Считать из файла пары значений и в тех из них, где X является точным квадратом Y или наоборот, найти сумму значений X иY. Результат записать в другой текстовый файл в виде X Y sum 5 25 30 1 3 49 7 56
9, 22	Случайным образом создать таблицу пар целочисленных значений и записать её в текстовый файл в виде: XY 5 1 2 8 12 3 - - - - Считать из файла пары значений и в тех из них, где Y кратен X, а X – четное, пометить строку таблицы: XY . 510 2 8 *** 12 3 - -

	в том же файле.
10, 23	Случайным образом создать таблицу пар значений и записать её в текстовый файл в виде: abc 5.2 4.6 2.5 можно 1.2 8.9 2.3 --- Считать из файла записанные данные и определить, можно ли построить треугольник с такими сторонами. В соответствующих строках (где можно), указать площадь полученного треугольника (в другом файле).
11, 24	Создать текстовый файл, содержащий целые значения, следующего формата 5 3 21 4 37 52 9 2 . . . Считать из файла записанные данные и определить минимальное значение. Если оно кратно трем, заменить каждое третье значение файла нулем, если кратно пяти – заменить его суммой первого и последнего элементов.
12, 25	Случайным образом создать таблицу пар значений и записать её в текстовый файл в виде: n * c 5 * m 7 * a 3 * q --- Преобразовать эту таблицу по следующему образцу (преобразования производить в исходном файле) n * c # 5 * m mmmmm 7 * a aaaaaaa 3 * qqqq --- Если первое значение не является числом, то в третьем столбце стоит один символ #
Номер варианта	Задание
13, 26	Ввести с клавиатуры значения вещественного типа и записать их в текстовый файл в виде таблицы следующего формата: X :Y : Z 2.1 : 3.7 : 0.9 6.2 : 5.4 : 4.2 --- - --- : --- Считать из файла полученные значения и создать из них другой файл вида: sin(max{X,Y,Z}) : cos(min{X,Y,Z}) значение sin(3.7) : значение cos(0.9) --- - ---

Контрольные вопросы

1. Каким образом можно открыть и закрыть файл для выполнения операций ввода-вывода в программе?
2. Как прочитать данные из файла в программу при помощи технологии программирования?

3. Как записать данные из программы в файл при помощи технологии программирования?
4. Как обработать ошибки при чтении или записи данных в файл при помощи технологии программирования?
5. Каким образом можно перемещаться по файлу и читать его данные порционно в программе?
6. Как реализовать отображение файла в память и работу с ним как с массивом данных в программе?
7. Как обрабатывать текстовые файлы и выполнять операции чтения и записи в них в программе?
8. Каким образом можно работать с двоичными файлами и производить операции чтения и записи бинарных данных в программе?
9. Как реализовать работу с файлами на удаленных серверах или в сети при помощи технологии программирования?
10. Как оценить производительность и эффективность операций ввода-вывода с файлами в программе и провести их оптимизацию?

Практическая работа №38
РАЗРАБОТКА МОДУЛЕЙ ЭКСПЕРТНОЙ СИСТЕМЫ

Цель работы: освоение технологии и методики построения экспертных систем на примере разработанной учебной экспертной системы

Порядок выполнения работы:

−выполнить задание;
−показать преподавателю;
−ответить на вопросы преподавателя.

Время выполнения: 2 ч

Теоретическая часть

Экспертная оболочка EsWin

ОБЩИЕ ПОЛОЖЕНИЯ

ESWinv. 1.0 - программная оболочка для работы с продукционно-фреймовыми экспертными системами с возможностью использования лингвистических переменных. Описываемая программная оболочка предназначена для решения задач методом обратного логического вывода на основе интерпретации правил-продукций с использованием фреймов как структур данных, включающих в себя в частности лингвистические переменные.

БАЗА ЗНАНИЙ

База знаний состоит из набора фреймов и правил-продукций. Формат внешнего представления базы знаний (в текстовом файле) выглядит следующим образом:

TITLE = <название экспертной системы>
COMPANY = <название предприятия>
FRAME // фрейм
<описание фрейма>

```
ENDF
  .
  .
  .
FRAME                          // фрейм
<описание фрейма>
ENDF
RULE                           // правило-продукция
<описание условий правила>
DO
<описание заключений правила>
ENDR
  .
  .
  .
RULE                           // правило-продукция
<описание условий правила>
DO
<описание заключений правила>
ENDR
```

База знаний состоит из двух частей: постоянной и переменной. Переменная часть базы знаний называется базой данных и состоит из фактов, полученных в результате логического вывода. Факты в базе данных не являются постоянными. Их количество и значение зависит от процесса и результатов логического вывода.

До начала работы с экспертной оболочкой база знаний находится в текстовом файле. В файле с расширением ***.klb** (**KnowLedgeBase**) хранятся фреймы и правила-продукции (база знаний). При начале работы с программной оболочкой наличие данного файла обязательно. Этот файл создается пользователем с помощью специального редактора или вручную. В файле с расширением ***.dtb** (**DaTaBase**) хранятся факты, полученные в процессе логического вывода (база данных). При начале работы с программной оболочкой наличие данного файла необязательно. Файл с базой данных создается программной оболочкой в процессе логического вывода. Первые части имен этих двух файлов совпадают.

При работе с программной оболочкой (после загрузки в оперативную память баз) фреймы и правила-продукции, находившиеся в файле с расширением ***.klb**, остаются неизменными. Факты, находившиеся в файле с расширением ***.dtb**, могут изменяться в процессе логического вывода (появляться, удаляться или менять свое значение в результате срабатывания правил-продукций или диалога с пользователем).

Пример базы знаний:

```
TITLE = для выбора метода представления знаний
FRAME = Цель
  Метод представления знаний: ()
ENDF
FRAME = Тип
  Решаемые задачи: (диагностика; проектирование)
ENDF
FRAME = Область
  Применение [Какова область применения?]: (медицина;
  вычислительная техника)
ENDF
FRAME = Действие
```

```
Сообщение: ()
ENDF

RULE 1
  = (Область.Применение; медицина)
  = (Тип.Решаемые задачи; диагностика)
DO
  = (Метод представления знаний; Правила-продукции с
    представлением нечетких знаний) 90
ENDR

RULE 2
  = (Область.Применение; вычислительная техника)
  = (Тип.Решаемые задачи; проектирование)
DO
  = (Метод представления знаний; Фреймы) 100
  = (Метод представления знаний; Правила-продукции с
    представлением нечетких знаний) 70
  = (Метод представления знаний; Семантические сети) 70
  MS (Действие.Сообщение; Доказано правило 4)
ENDR
```

ФРЕЙМЫ

Фреймы используются в базе знаний для описания объектов, событий, ситуаций, прочих понятий и взаимосвязей между ними. Фрейм - это структура данных, состоящая из слотов (полей). Формат внешнего представления фреймов (в текстовом файле) выглядит следующим образом:

```
FRAME (<тип фрейма>) = <имя фрейма>
PARENT: <имя фрейма-родителя>
OWNER: <имя фрейма-владельца>
<имя слота 1> (<тип слота>) [<вопрос слота>?]: (<значение 1>;
<значение 2>; ... ;
<значение k>)
<имя слота 2> (<тип слота>) [<вопрос слота>?]: (<значение 1>;
<значение 2>; ... ;
<значение l>)
  .
  .
  .
<имя слота n> (<тип слота>) [<вопрос слота>?]: (<значение 1>;
<значение 2>; ... ;
<значение m>)
ENDF
```

Фрейм может принадлежать к одному из трех типов фреймов: фрейм-класс (тип описывается зарезервированным словом "класс"), фрейм-шаблон (тип описывается зарезервированным словом "шаблон"), фрейм-экземпляр (тип описывается зарезервированным словом "экземпляр"). В базе знаний содержатся фреймы-классы и фреймы-шаблоны. При создании базы знаний тип фрейма-класса можно не описывать, этот тип фрейма понимается по умолчанию. Явно следует описывать только тип фрейма-шаблона.

В базе данных хранятся только фреймы-экземпляры. Так как для хранения фреймов-

экземпляров используется специальный файл с расширением *.dtb, явно их тип в этом файле также можно не описывать. (Описание типов фреймов-классов и фреймов-экземпляров используется по преимуществу во внутреннем представлении базы знаний и базы данных).

ИМЯ ФРЕЙМА, ФРЕЙМА-РОДИТЕЛЯ, ФРЕЙМА-ВЛАДЕЛЬЦА, СЛОТА

Имена фрейма, фрейма-родителя, фрейма-владельца, слота - это последовательность символов (русские и/или латинские буквы, цифры, пробелы, знаки подчеркивания).

Тип слота

Тип слота может принадлежать к одному из трех типов: символьный, численный, лингвистический. Описание типа слота определяет тип возможных значений слота. Обязательным является описание типов слотов численного (описывается зарезервированным словом "численный") и лингвистического (описывается зарезервированным словом "лп"). Слот без описания типа понимается как символьный по умолчанию.

Вопрос слота

Вопрос слота - любая последовательность символов. Вопрос слота не является обязательным. В таком случае, в процессе логического вывода, при возникновении необходимости задать вопрос пользователю, касающийся определения значения данного слота, пользователю будет предложена формулировка: "Выберите значение" или "Введите значение".

Значение слота

Значение слота - любая последовательность символов. Значения слота разделяются точками с запятыми. Список значений слота не обязателен, он может отсутствовать, в таком случае пустые круглые скобки необязательны. Во фрейме-экземпляре у каждого слота может быть только единственное значение, во фреймах-классах и фреймах-шаблонах число значений слотов не ограничено.

С помощью специальных слотов parent и owner фреймы могут объединяться в деревья. Кроме того, между фреймами могут существовать и произвольные связи через обычные слоты (значением слота в этом случае является имя другого фрейма).

Примеры фреймов:

FRAME = Цель
Метод представления знаний: ()
ENDF
FRAME = Тип
Решаемые задачи: (диагностика; проектирование)
ENDF
FRAME = Область
Применение [Какова область применения?]: (медицина; вычислительная техника)
ENDF
FRAME = Количество
Число правил в базе знаний (численный): ()
Число объектов в базе знаний (численный): ()
ENDF
FRAME = Действие
Сообщение: ()
ENDF

ПРАВИЛА-ПРОДУКЦИИ (ПРАВИЛА)

Правила используются в базе знаний для описания отношений между объектами, событиями, ситуациями и прочими понятиями. На основе отношений, задаваемых в правилах, выполняется логический вывод. В условиях и заключениях правил присутствуют ссылки на фреймы и их слоты. Формат внешнего представления правил (в

текстовом файле) выглядит следующим образом:
RULE <номер правила>
<условие 1>
<условие 2>
.
.
.
<условие m>
DO
<заключение 1>
<заключение 2>
.
.
.
<заключение n>
ENDR

Номер правила

Номер правила - целое число. Начало нумерации и порядок нумерации может быть произвольным, но из соображений целесообразности лучше нумеровать правила по порядку и начинать нумерацию с единицы.

Условие и заключение

Формат записи условий и заключений одинаков и имеет следующий вид:

<отношение> (<имя слота>; <значение слота>) <коэффициент достоверности>

Отношение

Отношения в условиях и заключениях могут быть EQ/= (равно), LT/< (меньше), GT/> (больше), EX (запуск внешней программы), MS (выдача сообщения), FR (вывод фрейма-экземпляра). В заключениях правил используются только отношения EQ/= (равно), EX (запуск внешней программы), MS (выдача сообщения) и FR (вывод фрейма-экземпляра). Для строковых значений слотов могут использоваться только отношения EQ/= (равно), EX (запуск внешней программы), MS (выдача сообщения), FR (вывод фрейма-экземпляра). Для слотов лингвистического типа допустимы все отношения, так как с ними связаны как строковые, так и численные значения.

Имя слота

Имя слота может быть локальным или глобальным. Локальное имя слота - имя, соответствующее имени слота в некотором фрейме. Глобальное имя слота - имя фрейма, которому принадлежит слот и собственно имя слота, разделенные точкой.

Пример локального имени слота: *Применение*

Пример глобального имени слота: *Область.Применение*

Значение слота

Значение слота - строка или число, в зависимости от типа слота. Если в качестве значения слота используется имя фрейма-шаблона, то в процессе логического вывода выполняется одновременное определение значений для всех слотов данного фрейма.

Коэффициент достоверности

Коэффициент достоверности - число от 0 до 100. Коэффициент достоверности в заключении используется при формировании значения слота фрейма-экземпляра при срабатывании правила.

Примеры правил:

RULE 1
= (Область.Применение; медицина)
= (Тип.Решаемые задачи; диагностика)
DO

= (Метод представления знаний; Правила-продукции с представлением нечетких знаний) 90
ENDR
RULE 2
= (Область.Применение; вычислительная техника)
= (Тип.Решаемые задачи; проектирование)
DO
= (Метод представления знаний; Фреймы) 100
= (Метод представления знаний; Правила-продукции с представлением нечетких знаний) 70
= (Метод представления знаний; Семантические сети) 70
MS (Действие.Сообщение; Доказано правило 4)
ENDR

ИНТЕРПРЕТАЦИЯ ПРАВИЛ-ПРОДУКЦИЙ

Интерпретация правил начинается с выбора цели логического вывода. В качестве цели логического вывода используются целевые слоты, содержащиеся во фрейме-классе со специальным именем "Цель".

Далее определяется правило, в заключении которого присутствует выбранный целевой слот.

После определения правила начинается его интерпретация (перебор и проверка условий). При проверке условия ищется соответствующий слот. Первоначальный поиск выполняется в базе данных. Если слот имеет значение, то оно используется при проверке условия. Если значения нет, то значение слота запрашивается у пользователя, с использованием меню выбора символьных значений, или окна для ввода численного значения, или того и другого в случае слота лингвистического типа. Слот в условии может указываться своим локальным именем или глобальным (с указанием имени фреймов). При локальном имени слота поиск начинается с фрейма, использованного последним при логическом выводе. Такой фрейм считается текущим. Имя текущего фрейма хранится в качестве значения слота специального фрейма, описывающего контекст диалога. Этот фрейм всегда доступен для проверки условия в правилах.

При вводе пользователем значения слота лингвистического типа, формируется численное значение с коэффициентом достоверности равным 100, если пользователь ввел число, если пользователь выбрал символьное значение, формируется символьное значение с коэффициентом достоверности равным 100. Если значение слота в правиле было символьным, а пользователем было введено численное значение, то коэффициент достоверности формируется как значение функции принадлежности лингвистической переменной (введенное пользователем число используется в качестве аргумента функции принадлежности).

Коэффициент достоверности набора условий вычисляется как коэффициент достоверности конъюнкции (минимальное значение из значений коэффициентов достоверности условий).

Коэффициент достоверности слота фрейма-экземпляра, формируемого на основе заключения, вычисляется как произведение коэффициент достоверности набора условий и коэффициента достоверности заключения. Если такой слот во фрейме-экземпляре уже есть, то его коэффициент достоверности меняется на новое значение, вычисляемое по формуле:

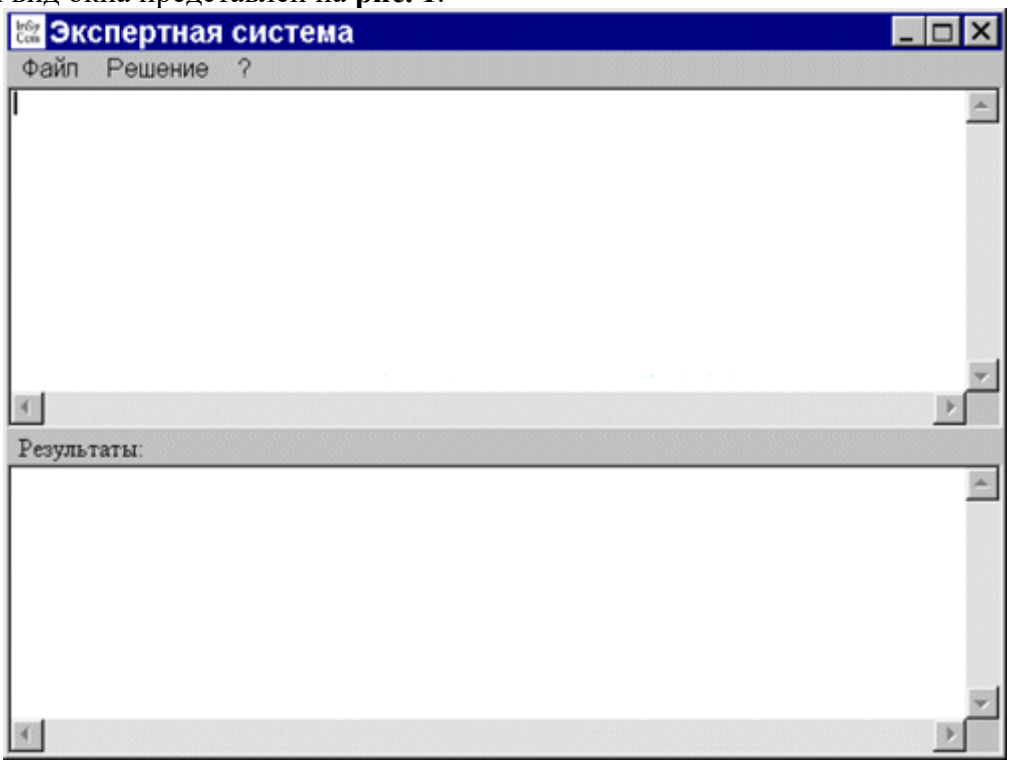
$$КД_{результирующий} = КД_{исходного\ слота} + КД_{набора\ условий} * (1-КД_{исходного\ слота})$$

Рабочее задание

Выполнить инструкции приведенные ниже

ПОРЯДОК РАБОТЫ С ПРОГРАММНОЙ ОБОЛОЧКОЙ

Исполняемый модуль программной оболочки находится в файле **ESWin.exe**.
Общий вид окна представлен на **рис. 1**.



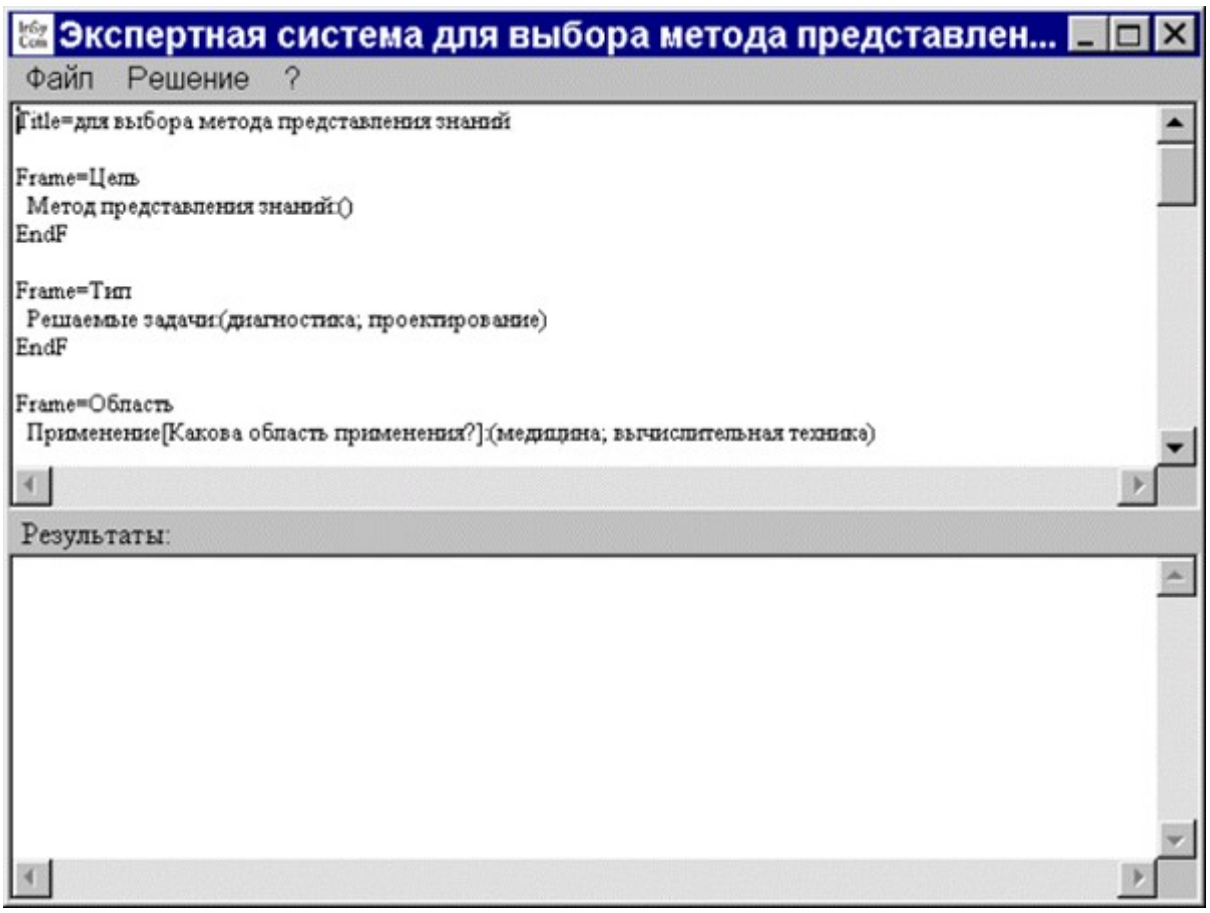
В строке заголовка окна при работе с конкретной базой знаний (экспертной системой) выводится название экспертной системы (строка, задаваемая в базе знаний зарезервированным словом **TITLE**).

Строка меню состоит из пунктов: "**Файл**", "**Решение**" и "**?**".

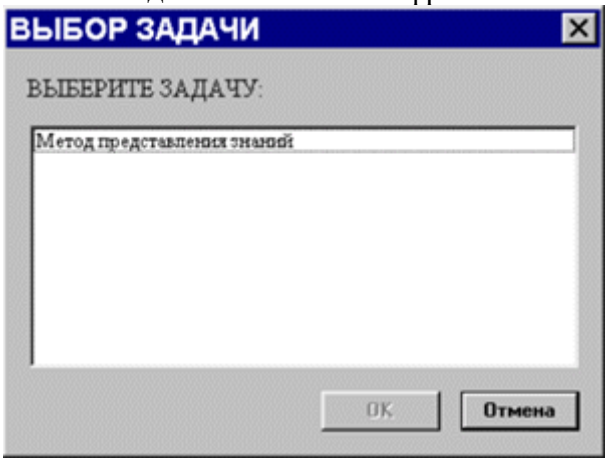
Работа с конкретной базой знаний начинается с ее загрузки. Для этого используется пункт меню "**Файл**"/"**Загрузить базу знаний**". База знаний находится в файле расширением ***.klb**. Если в загруженной базе знаний во фреймах-классах используются слоты лингвистического типа, то файл с описанием лингвистических переменных загружается автоматически.

При необходимости можно загрузить базу данных из одноименного файла с расширением ***.dtb**. Для этого используется пункт меню "**Файл**"/"**Загрузить базу данных**".

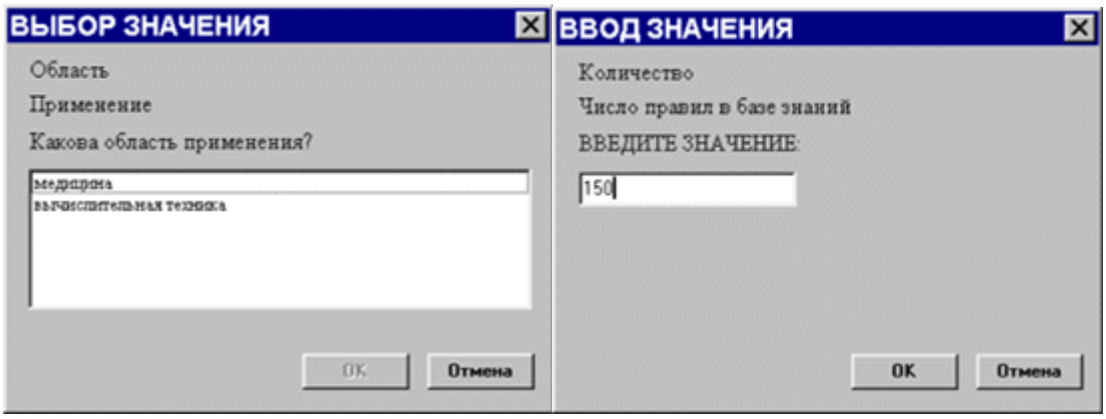
После загрузки фреймы и правила базы знаний отображаются в верхней части основного окна (**рис. 2**)



Для начала логического вывода используется пункт меню "**Решение**"/"**Поиск решения**". После выбора этого пункта меню на экране появляется окно "**Выбор задачи**" с перечнем целей логического вывода, одну из которых требуется выбрать (**рис. 3**). Перечень целей логического вывода описывается во фрейме-классе с именем "**Цель**".

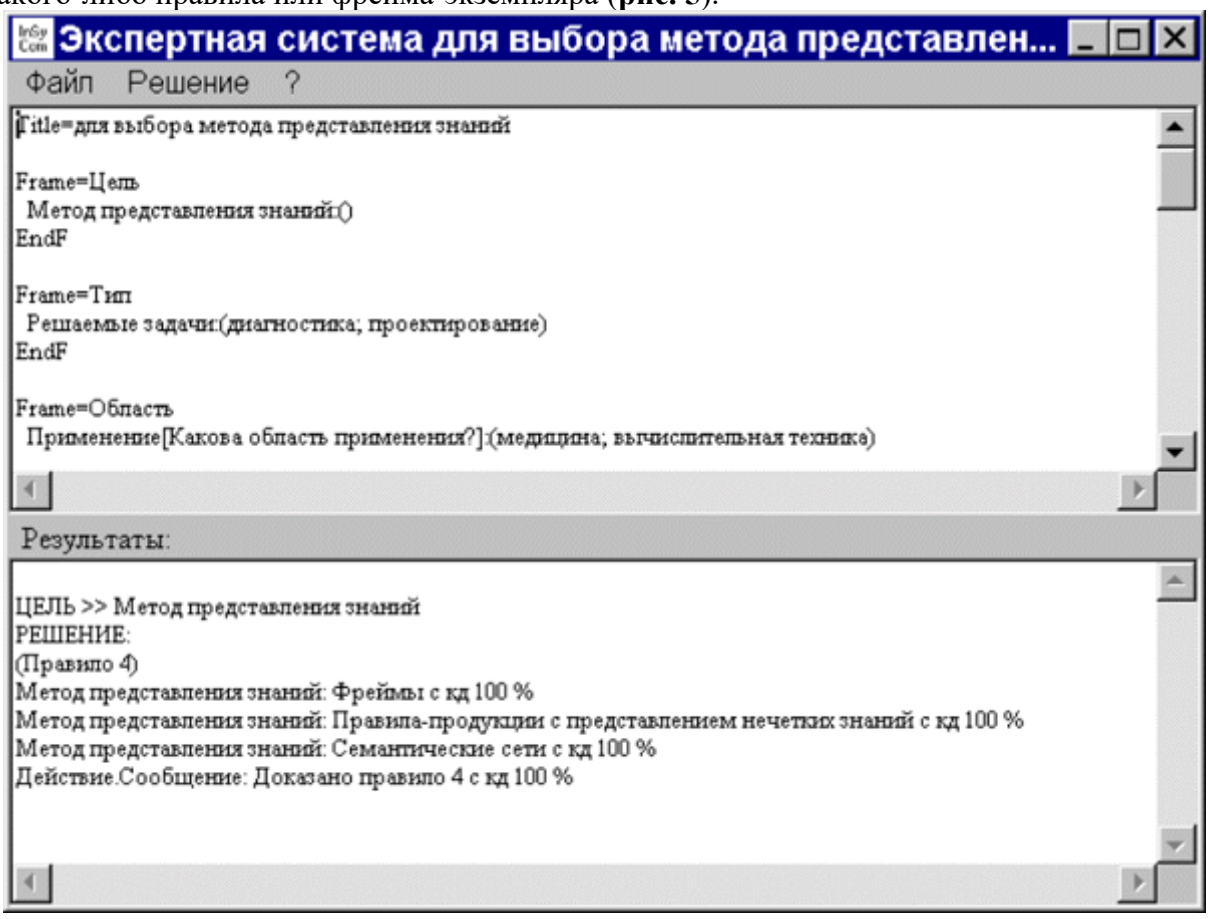


В процессе логического вывода, в качестве ответов на вопросы, задаваемые программной оболочкой, пользователю предлагается выбирать одно из символьных значений или вводить численное значение (**рис. 4**).

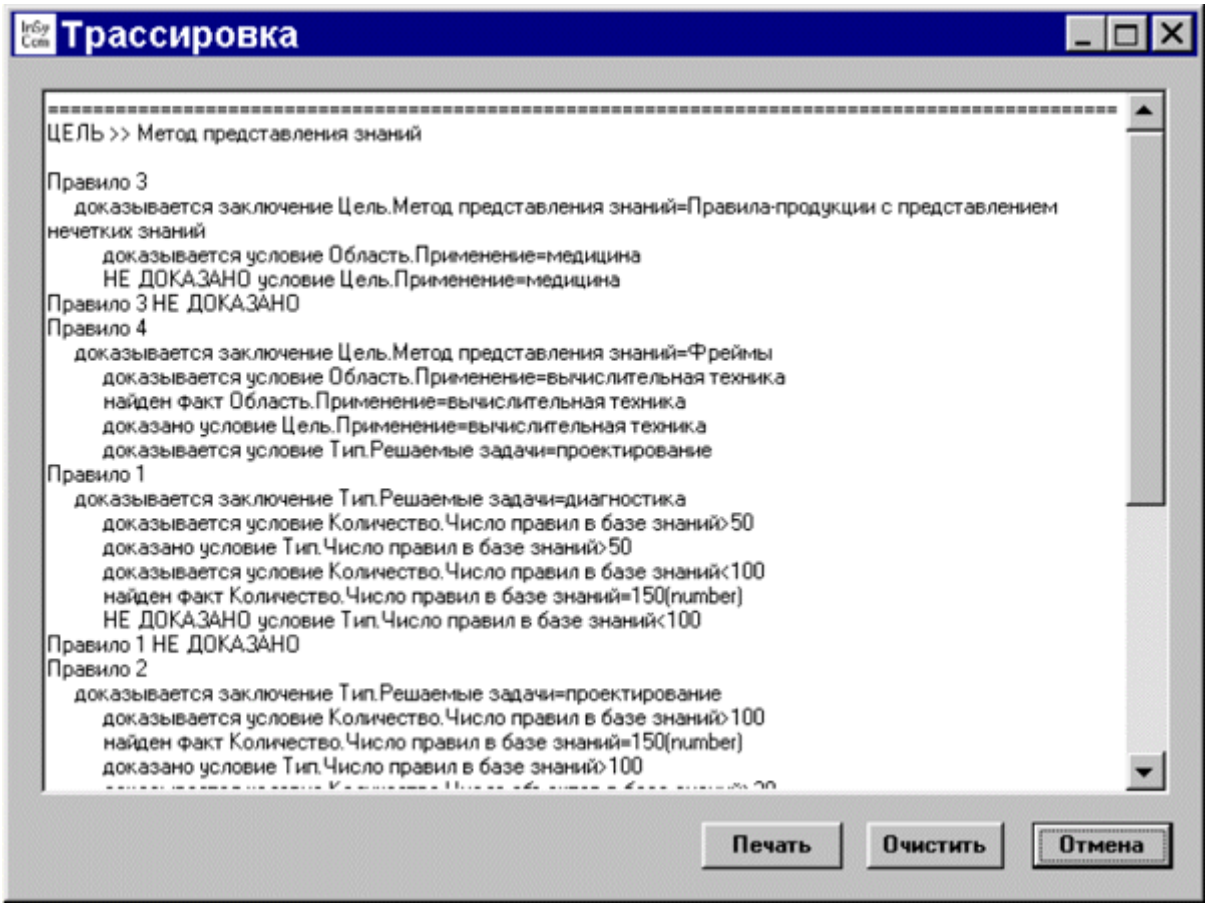


В случае с лингвистической переменной в одном окне будет предложено выбрать одно из символьных значений или ввести численное значение.

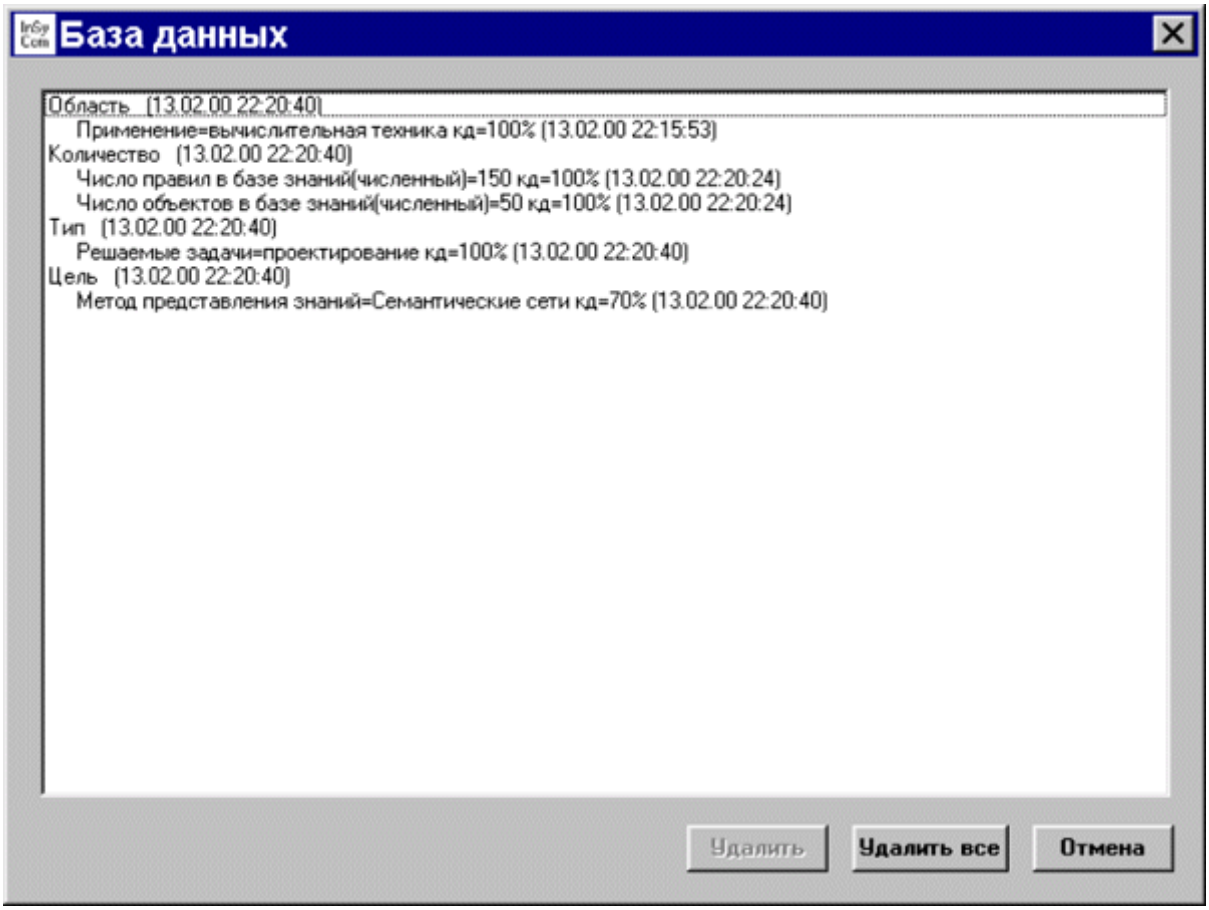
Результаты логического вывода отображаются в нижней части основного окна с комментариями, каким образом было получено решение: в результате доказательства какого-либо правила или фрейма-экземпляра (**рис. 5**).



Для просмотра последовательности шагов, выполненных программной оболочкой в процессе логического вывода, можно воспользоваться пунктом меню "**Решение**"/"**Трассировка**". При необходимости можно распечатать или удалить результаты трассировки (**рис. 6**).



Для просмотра фреймов-экземпляров, полученных в результате вывода можно воспользоваться пунктом меню "**Решение**"/"**Просмотр базы данных**" (**рис. 7**) или просмотреть содержимое файла с расширением ***.dtb** (этот файл постоянно обновляется в процессе логического вывода). При необходимости можно удалить отдельный слот во фрейме-экземпляре, полностью фрейм-экземпляр, все фреймы-экземпляры. Все эти изменения сразу же сохраняются в файле с расширением ***.dtb**.



Пункт меню "**Решение**"/"**Очистка базы данных**" используется для удаления всех фреймов-экземпляров из загруженной базы данных. То же действие можно проделать и с помощью пункта меню "**Решение**"/"**Просмотр базы данных**" (кнопка "**Удалить все**").

Пункты меню "**?**"/"**Вызов справки**" и "**?**"/"**О программе**" используются для получения справочной информации и сведений о программе.

Для завершения работы с программной оболочкой **ESWin** используется пункт меню "**Файл**"/"**Выход**".

Семантические сети

Термин семантическая означает смысловая, а сама семантика - это наука, устанавливающая отношения между символами и объектами, которые они обозначают, т.е. наука, определяющая смысл знаков,

Семантическая сеть- это ориентированный граф, вершины которого - понятия, а дуги - отношения между ними.

Понятиями обычно выступают абстрактные или конкретные объекты, а отношения - это связи типа: "это" ("is"), "имеет частью" ("haspart"), "принадлежит", "любит". Характерной особенностью семантических сетей является обязательное наличие трех типов отношений:

класс - элемент класса;

свойство - значение;

пример элемента класса.

Можно ввести несколько классификаций семантических сетей. Например, по **количеству типов отношений**:

однородные (с единственным типом отношений);

неоднородные (с различными типами отношений).

По типам отношений:

бинарные (в которых отношения связывают два объекта);

парные (в которых есть специальные отношения, связывающие более двух понятий).

Наиболее часто в семантических сетях используются следующие отношения:
связи типа "часть-целое" ("класс-подкласс", "элемент-множество" и т.п.);
функциональные связи (определяемые обычно глаголами "производит", "влияет"...);
количественные (больше, меньше, равно...);
пространственные (далеко от, близко от, за, под, над...);
временные (раньше, позже, в течение...);
атрибутивные связи (иметь свойство, иметь значение...);
логические связи (и, или, не) и др.

Проблема поиска решения в базе знаний типа семантической сети сводится к задаче поиска фрагмента сети, соответствующего некоторой подсети, соответствующей поставленному вопросу.

Пример. На рисунке изображена семантическая сеть. В качестве вершин понятия: Человек, Иванов, Волга. Автомобиль, Вид транспорта. Двигатель.

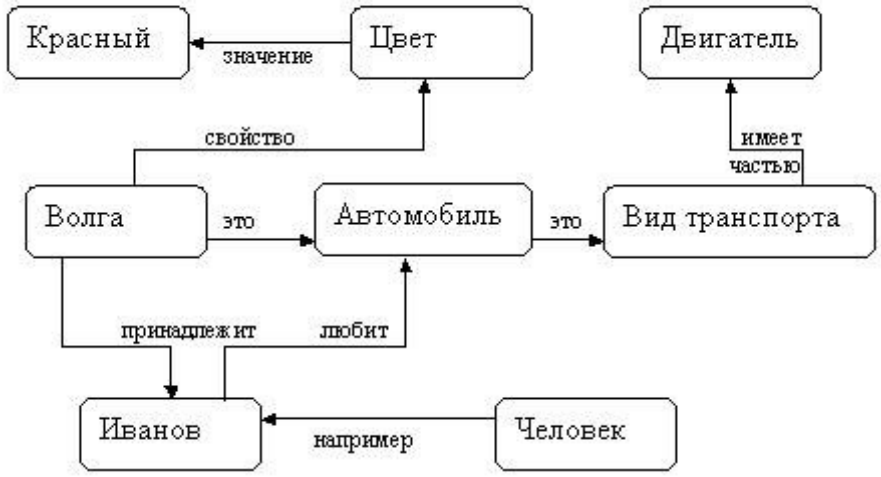


Рис. Семантическая сеть.

Основное преимущество этой модели - в соответствии современным представлениям об организации долговременной памяти человека.

Недостаток модели - сложность поиска вывода на семантической сети.
Для реализации семантических сетей существуют специальные сетевые языки, например NET и др. Широко известны экспертные системы, использующие семантические сети в качестве языка представления знаний - PROSPECTOR, CASNBT, TORUS.
При формализации созданной семантической сети как правило используется ее матричное (или табличное) отображение.

Контрольные вопросы

1. Как выбрать подходящую технологию для разработки учебной экспертной системы?
2. Каким образом была спроектирована структура и архитектура учебной экспертной системы?
3. Какой метод или подход к инженерии знаний был использован при разработке учебной экспертной системы?
4. Какое базовое знание и правила были определены для учебной экспертной системы?
5. Каким образом были собраны и структурированы данные для обучения учебной экспертной системы?
6. Как проводилась верификация и валидация учебной экспертной системы?
7. Каким образом пользователи могли взаимодействовать с учебной экспертной системой?
8. Как реализована система вывода и интерпретации результатов в учебной экспертной системе?

9. Как реализован механизм обновления и поддержания знаний в учебной экспертной системе?
10. Как можно оценить эффективность и практическую ценность учебной экспертной системы и провести ее апробацию в реальной среде?

Практическая работа №39
СОЗДАНИЕ СЕТЕВОГО СЕРВЕРА И СЕТЕВОГО КЛИЕНТА

Цель работы: Изучение протоколов семейства TCP/IP, функций и методов стандарта WINSOCK, определяющего сетевой интерфейс для программирования сокетов в ОС MicrosoftWindows. Разработка программы-клиента в архитектуре взаимодействия “клиент-сервер” с использованием семейства протоколов TCP/IP.Разработка программы-сервера в архитектуре взаимодействия “клиент-сервер” с использованием семейства протоколов TCP/IP и библиотеки WinSock.

Порядок выполнения работы:
−выполнить задание;
−показать преподавателю;
−ответить на вопросы преподавателя.

Время выполнения: 2 ч

Рабочее задание: Разработать программу клиента, которая должна:
- запрашивать у пользователя адрес программы-сервера;
- устанавливать соединение с сервером;
- передавать на сервер данные;
- принимать ответ от сервера и выводить его на экран;
- закрывать соединение с сервером.

Теоретическая часть

Для того чтобы установить связь с процессом, выполняющимся на другой ЭВМ, клиентская программа должна создать сокет. Сокет в любой современной системе представляет собой особый вид файла, из которого можно читать и в которой можно записывать двоичные данные. При операциях обмена с сокетом нет никакого контроля типов, эта задача возлагается на приложения. На схеме алгоритма (см. Error: Reference source not found) блок “Создание сокета” подразумевает вызов функции socket, который в случае успеха создает сокет – потоковое или дейтаграммное – и семейство протоколов. В данном случае создается потоковый сокет и используется семейство протоколов TCP/IP.

Установка соединения с другим процессом заключается в обмене специальными пакетами и возможно только тогда, когда тот процесс ожидает приема соединений. В противном случае результат операции будет неудачным и будет получено сообщение о том, что-либо не удалось установить соединение, либо оно было разорвано (зависит от реализации). Для установки соединения требуется указать ЭВМ по IP-адресу или по доменному имени, которое обязательно должно быть преобразовано в IP-адрес, и процесс на этой ЭВМ (по целочисленному идентификатору, называемому портом). Все это реализуется при помощи функции connect. Если соединение успешно установлено, то сразу после вызова этой функции можно вести обмен с гарантированной доставкой пакетов. В противном случае работа невозможна.

Передача и прием данных, то есть обмен с сокетом, производится всеми доступными в системе средствами обмена с файлами, например, системные вызовы read и write в UNIX и Windows, библиотечные функции fprintf, fgets и т. д. Перед осуществлением передачи данные, если это

необходимо, шифруются каким-либо алгоритмом. На принимающей стороне полученные данные дешифруются, и определяется (или опровергается) их подлинность, от результата чего зависит дальнейшая работа с этим клиентом.
Разрыв соединения означает обмен специальными пакетами и может производиться при помощи системного вызова close. Функция close уничтожает сокет, делая его непригодным к использованию (любая операция с ним будет заканчиваться неудачей).
Задание 2: Разработать программу сервера, которая должна:

- ожидать запросов от программ клиентов на соединение;
- устанавливать соединение с клиентами; принимать данные от клиентов и выполнять их обработку;
- пересылать результат обработки клиенту.

Основные сведения:
Основной задачей серверной части является обработка. Обмен данными с клиентскими процессами есть важная составляющая часть этой задачи.
Для того чтобы процессы-клиенты могли связаться с сервером, сервер создает сокет для обмена данными. На схеме алгоритма (см. Error: Reference source not found) это представлено блоком "Создание сокета". Производится так же, как и в клиентской программе.
Следующий блок – "Получение локального адреса" – принципиально важен. Он служит для того, чтобы все запросы на соединения, приходящие на данную ЭВМ и обращающиеся к указанному порту, операционная система направляла данному процессу. Операция производится посредством системного вызова bind, в котором указывается созданный ранее сокет, IP–адрес ЭВМ (как правило, это константа 0) и идентификатор процесса, т. е. порт. После этого, в случае успеха, программа сервера вызывает функцию listen, которая говорит операционной системе о том, что процесс ожидает поступления запросов на соединение к данному сокету и, что эти запросы нужно ставить в очередь указанной длины (в штуках).
Получение запроса на соединение происходит тогда, когда клиентский процесс вошел в блок "Установка соединения", т. е. вызвал функцию connect. ОС сервера при этом создает копию сокета, чтобы программа могла на первом экземпляре продолжить работу, а на другом – вести обмен с подключившимся клиентом. Следующий блок – "Создание нового потока" – подразумевает порождения новой параллельной ветки программы, которая будет вести обработку данных. В системах Windows это обычно нить (thread), создаваемая при помощи функции _beginthread, в UNIX-системах это новый процесс, создаваемый при помощи вызова fork.
Передача и прием данных, т. е. обмен с сокетом, производится всеми доступными в системе средствами обмена с файлами, например: системные вызовы read и write в UNIX и Windows, библиотечные функции fprintf, fgets и т. д.После приема данных они дешифруются с целью, во-первых, получить открытый текст и, во-вторых, чтобы определить подлинность данных. Затем если установлена подлинность данных и получен корректный открытый текст, производится обработка данных. Перед отправкой клиенту результатов работы данные снова шифруются.

Контрольные вопросы

1. Что представляет собой семейство протоколов TCP/IP и какие протоколы оно включает?
2. Каким образом устанавливается и разрывается TCP-соединение между клиентом и сервером?
3. Каким образом гарантируется доставка данных в протоколе TCP?
4. Какие функции и методы предоставляет стандарт WINSOCK для программирования сокетов в ОС?
5. Как реализовать создание сокета и его связь с конкретным портом при помощи стандарта WINSOCK?
6. Как реализовать отправку и прием данных через сокет при помощи стандарта WINSOCK?
7. Каким образом обрабатывать ошибки и исключения, возникающие при работе с сокетами в стандарте WINSOCK?

8. Каким образом реализована мультиплексирование и обработка нескольких сокетов одновременно при помощи стандарта WINSOCK?
9. Каким образом управлять уровнем использования пропускной способности и настройками сокетов при помощи стандарта WINSOCK?
10. Как оценить производительность и эффективность работы с сокетами при помощи стандарта WINSOCK и провести оптимизацию сетевого взаимодействия в программе?

Перечень использованных информационных ресурсов

№	ISBN	Название	Назначение	Автор	Издательство	Год издания	Издание	Кол-во в библиотеке	Объем	Наличие на электронных носителях
1		2	3	4	5	6	7	8	9	10
	978-5-4486-0513-0	Объектно-ориентированное программирование и программная инженерия		Мейер Б.	Москва: Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Эр Медиа	2019	Объектно-ориентированное программирование и программная инженерия	1	285 с.	http://www.iprbookshop.ru/79706.html
	978-5-4468-6992-3	Разработка модулей программного обеспечения для компьютерных систем	учебник для студ. СПО	Федорова Г.Н.	Академия	2018	2-е изд, стер.	25	384 с.	
	978-5-4468-6739-4	Разработка и эксплуатация автоматизированных информационных систем	учеб. пособие для студ. СПО	Фуфаев Д.Э.	Академия	2018	6-е изд.,стер.	25	304 с.	
	978-5-4488-0101-3	Алгоритмы и структуры данных	Электрон.текстовые данные (электронный ресурс)	Никлаус, Вирт	Саратов:Профобразование	2017		1	272 с.	http://www.iprbookshop.ru/63821.html
	978-5-4468-6169-9	Основы алгоритмизации и программирования. Практикум	учеб. пособие для сред. проф. образования	Семакин ,И.Г.	М.:Академия	2018	2-е изд.	25	144 с.	