Министерство образования Белгородской области
Областное государственное автономное профессиональное образовательное учреждение
«Белгородский индустриальный колледж»

Рассмотрено
предметно-цикловой комиссией
протокол заседания № 1
от «31» августа 2022г.
Председатель цикловой комиссии
_____/ Третьяк И.Ю.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

МДК 05.02 Разработка кода информационных систем

по специальности
09.02.07 Информационные системы и программирование

квалификация
Разработчик веб и мультимедийных приложений

Разработчик:
Преподаватель ОГАПОУ «БИК»
Внукова Н.В.

Белгород 2022 г.

Пояснительная записка

МДК 05.02 «Разработка кода информационных систем» входит в состав ПМ.05Проектирование и разработка информационных систем, формирует базовые умения для освоения выпускником профессиональных компетенций.

Методические указания по выполнению лабораторных работ составлены в соответствии с рабочей программой профессионального модуля ПМ.05 Проектирование и разработка информационных систем, МДК 05.02. «Разработка кода информационных систем» для специальности 09.02.07Информационные системы и программирование и рассчитаны на 52 часа.

Целью методических указаний по выполнению лабораторных работ является организация и управление работой обучающихся на лабораторных занятиях при изучении МДК05.02. «Разработка кода информационных систем».

Методические указания по выполнению лабораторных работ содержат общие положения, тематику лабораторных работ, содержание лабораторных работ и требования к оформлению отчетов.

Методические указания к каждой лабораторной работе включают в себя следующие элементы: название темы, цель занятия, ход работы, теоретическую часть, практическую часть (указания по выполнению) и контрольные вопросы.

Методические указания содержат лабораторные работы, которые обеспечивают формирование базовых умений и навыков владения основными методологиями процессов разработки программного обеспечения, использования методов для получения кода с заданной функциональностью и степенью качества, раскрывают принципы построения, структуры и приемы работы с инструментальными средствами, поддерживающими создание программного обеспечения.

В лабораторных работах, приведенных в данных методических рекомендациях, содержатся как задания с подробными указаниями к выполнению, так и задания без алгоритма работы.

Методические рекомендации предназначены для студентов очной формы обучения специальности 09.02.07 «Проектирование и разработка информационных систем».

Методические рекомендации направлены на повышение мотивации учащихся к изучению МДК 05.02 «Разработка кода информационных систем», развитие гибкого логического и пространственного мышления обучающихся, развитие их профессиональных компетенций.

Тематика лабораторных работ

Номер лабораторной работы	Тема лабораторной работы	Кол-во часов
1	Case-средства для моделирования деловых процессов	2
2	Работа в инструментальной среде	2
3	Создание контекстной диаграммы IDEF0	2
4	Моделирование бизнес-процессов с помощью инструментальных средств	2
5	Отображение модели данных в инструментальном средстве	2
6	Построение диаграммы Вариантов использования и диаграммы Последовательности и генерация кода	2
7	Построение диаграммы Кооперации и диаграммы Развертывания и генерация кода	2
8	Построение диаграммы Деятельности, диаграммы Состояний и диаграммы Классов и генерация кода	2
9	Построение диаграммы компонентов и генерация кода	2
10	Построение диаграмм потоков данных и генерация кода	2
11	Построение диаграмм потоков данных и генерация кода	2
12	Стоимостная оценка проекта	2
13	Построение и обоснование модели проекта	2
14	Установка и настройка системы контроля версий с разграничением ролей	2
15	Проектирование и разработка интерфейса пользователя	2
16	Разработка графического интерфейса пользователя	2
17	Реализация алгоритмов обработки числовых данных. Отладка приложения	2
18	Реализация алгоритмов поиска. Отладка приложения	2
19	Реализация обработки табличных данных. Отладка приложения	2

20	Разработка и отладка генератора случайных символов	2
21	Разработка приложений для моделирования процессов и явлений. Отладка приложения	2
22	Интеграция модуля в информационную систему	2
23	Программирование обмена сообщениями между модулями	2
24	Организация файлового ввода-вывода данных	2
25	Разработка модулей экспертной системы	2
26	Организация файлового ввода-вывода данных	2
ИТОГО:		**52**

Лабораторная работа № 1

Тема «Case-средства для моделирования деловых процессов»

Цель работы: ознакомиться с инструментальными средствами моделирования деловых процессов, их классификацией и примерами.

Теоретическая часть

Все CASE-средства делятся на типы, категории и уровни.

Классификация по типам отражает функциональную ориентацию CASE-средств в технологическом процессе.

1) Анализ и проектирование. Средства данной группы используютсядля создания спецификаций системы и ее проектирования; они поддерживаютшироко известные методологии проектирования. Ктакимсредствамотносятся: CASE, Аналитик (Эйтэкс), The Developer (ASYST Technologies),POSE (Computer Systems Advisers), ProKil*Workbench (McDonnell Douglas),Excelerator (Index Technology), Design-Aid (Naslec), Design Machine (Optima),MicroStep (Meta Systems), vsDesigner (Visucil Sofhvctrr), Anuijsl'Designer(Yourdon), Design/lDEF (Meta Soft\vare), BPWin (Logic Works). SELECT(Select Sofhvare Tools), System Architect (Popkin Software & Systems),Wesfmounl I-C.iSE Youi-don (Westmount Technology B. V. & CADRETechnologies), CASE/4/0 (microTOOL GmbH). Их целью является определениесистемных требований и свойств, которыми система должна обладать, атакже создание проекта системы, удовлетворяющей этим требованиям иобладающей соответствующими свойствами. На выходе продуцируютсяспецификации компонент системы и интерфейсов, связывающих этикомпоненты, а также "калька" архитектуры системы и детальная "калька"проекта, включающая алгоритмы и определения структур данных.

2) Проектирование баз данных и файлов. Средства данной группыобеспечивают логическое моделирование данных, автоматическоепреобразование моделей данных в Третью Нормальную Форму,автоматическую генерацию схем БД и описаний форматов файлов на уровнепрограммногокода: ERWin (LogicWorks), ChenToolkit (Chen&Asssociales),S-Designor (SDP), Designer2000 (Oracle), Silverrun (ComputerSystemsAdvisers).

3) Программирование Средства этой группы поддерживают этапыпрограммирования и тестирования, а также автоматическую кодогенерациюиз спецификаций, получая полностью документированную выполняемуюпрограмму: COBOL 2/Workhench (MikroFocus), DECASE (DEC),NETRON/CAP (Netron), APS (SageSoftwtire). Помимо диаграммеровразличного назначения и средств поддержки работы с репозиторием.в этугруппу средств включены и традиционные генераторы кодов, анализаторыкодов (как в статике, так и в динамике), генераторы наборов тестов,анализаторы покрытия тестами, отладчики.

4) Сопровождение и реинжиниринг. К таким средствам относятсядокументаторы, анализаторы программ, средства реструктурирования иреинжениринга: AdpacCASETools (Adpac).Scan/COBOL, uSuperstructure(Computer Data Systems), Inspector/Recoder (Language Technology).Их цельюявляется корректировка, изменение, анализ, преобразование

и реинженирингсуществующей системы. Средства позволяют осуществлять поддержку всейсистемной документации, включая коды, спецификации, наборы тестов;контролировать покрытие тестами для оценки полноты тестируемости;управлять функционированием системы и т.п. Особый интерес представляютсредства обеспечения мобильности (в CASE они получили название средствмиграции) и реинжиниринга. К средствам миграции относятся трансляторы,конверторы, макрогенераторы и др., позволяющие обеспечить переносуществующей системы в новое операционное или аппаратурное окружение.

Практическая часть

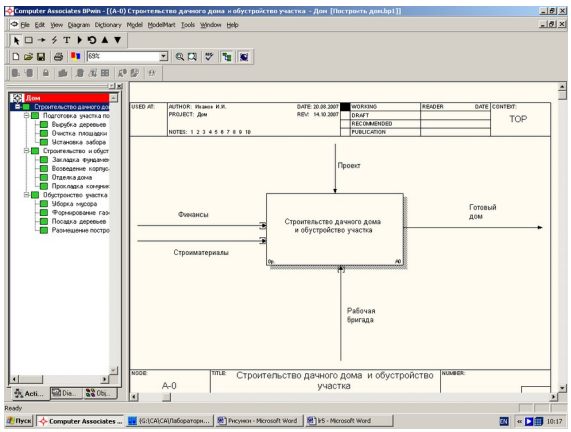
1. Изучите интерфейс программного продукта BPwin
2. Опишите элементы управления основной панели инструментов BPwin
3. Создайте новую модель.
4. Разработайте контекстную страницу модели.
5. Обдумайте, на какие функции может быть разложена главная функция системы, обозначенная Вами в функциональном блоке на контекстной странице модели. Помните, что число этих функций должно быть от 3 до 6.
6. Создайте диаграмму декомпозиии первого уровня. При создании диаграммы выберите в диалоговом окне нотацию диаграммы (IDEF0) и укажите, сколько функциональных блоков вы планируете разместить на диаграмме.
7. На диаграмме декомпозиции впишите названия выделенных функций в функциональные блоки. Помните о том, что функциональные блоки на диагонали должны быть расположены в порядке убывания их значимости или в соответствии с последовательностью выполнения работ.
8. Соедините интефейсные дуги, которые мигрировали с диаграммы верхнего уровня на созданную диаграмму декомпозиции в виде стрелок, с функциональными блоками в соответствии с их назначением.
9. Если в этом есть необходимость, сделайте разветвления дуг. Помните о том, что Вы можете оставить единое название для всех веток. В этом случае название располагается до разветвления стрелки. В случае, если ветки обозначают разные объекты, подпишите каждую ветку.
10. Создайте внутренние дуги, связывающие функциональные блоки между собой. Помните, что каждый функциональный блок обязательно должен иметь дуги Управления и Выхода. Дуги Механизма и Входа могут отсутствовать. Именуйте каждую дугу.
11. По описанной выше технологии создайте диаграммы декомпозиции для тех функциональных блоков, прояснить содержание которых требуется по логике модели.

Желаемый результат:

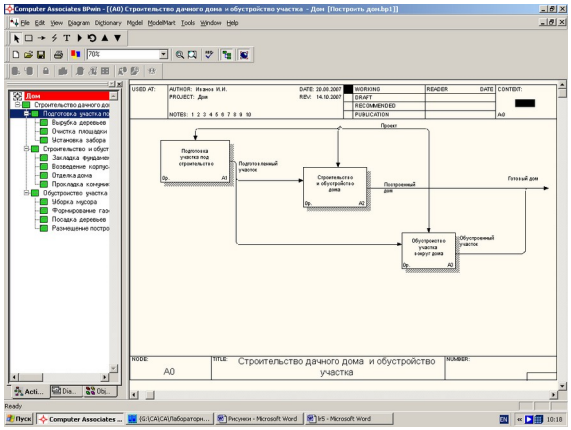
Для моделируемой системы в среде BPwin должна быть создана трехуровневая функциональная модель, содержащая кроме контекстной диаграммы, диаграммы двух уровней декомпозиции.

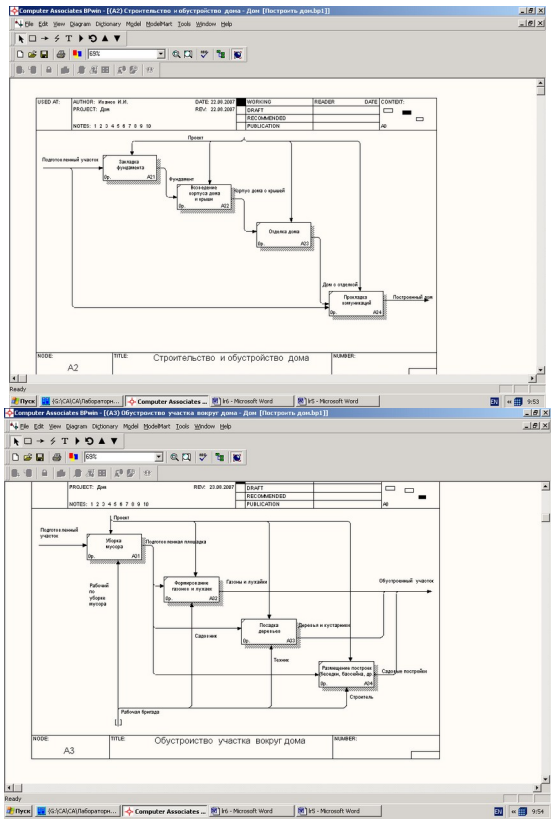
Действия:

1. Создадим новую модель.
2. Разработаем диаграмму верхнего уровня модели (контекстную).



3. Определим функции, на которые может быть разложена функция, обозначенная на контекстной странице модели. Это:
   - подготовка участка под строительство;
   - строительство и обустройство дома;
   - обустройство участка.
4. Создадим диаграмму декомпозиции первого уровня. Для этого:
   - выделим функциональный блок на контекстной странице;
   - на панели инструментов щелкнем по кнопке с изображением черного треугольника, направленного вершиной вниз (декомпозиция)
   - в диалоговом окне укажем нотацию создаваемой диаграммы (IDEF0) и число функциональных блоков, которые она должна содержать (3 - по числу выделенных функций).
5. На диаграмме декомпозиции впишем названия выделенных функций в функциональные блоки.
6. Соединим с функциональными блоками интефейсные дуги, которые мигрировали на созданную диаграмму декомпозиции с контекстной диаграммы.
7. Создадим внутренние дуги для связи функциональных блоков между собой.
8. Аналогично создадим диаграммы декомпозиции для функциональных блоков A1, A2, A3.





Контрольные вопросы

1. Для чего используется методология IDEF0.
2. Объясните необходимость задания цели и точки зрения модели?
3. Перечислите и расскажите назначения кнопок на панели инструментов.
4. Перечислите этапы декомпозиции блока.
5. Расскажите, каким образом на диаграмму добавить блок, дугу.

Лабораторная работа № 2

Тема «Работа в инструментальной среде»

Цель работы: ознакомиться с инструментальными средствами моделирования деловых процессов.

Теоретическая часть

Rational Rose — мощный инструмент анализа и проектирования объектно-ориентированных программных систем. Он позволяет моделировать системы до написания кода, так что вы можете с самого начала быть уверены в адекватности их архитектуры. С помощью готовой модели недостатки проекта легко обнаружить на стадии, когда их исправление не требует еще значительных затрат.

Среда Rational Rose позволяет проектировать варианты использования и их диаграммы для визуализации функциональных возможностей системы. Диаграммы Взаимодействия показывают, как объекты работают совместно, обеспечивая требуемые функциональные возможности. Для отображения объектов системы и их отношений используются диаграммы Классов. Диаграммы Компонентов иллюстрируют, как классы соотносятся с готовыми физическими компонентами системы. Наконец диаграммы Размещения применяют для визуализации проекта распределенных систем.

Модель Rose — это картина системы. Она содержит все диаграммы UML, действующих лиц, варианты использования, объекты, классы, компоненты и узлы системы. Она детально описывает, что система содержит и как функционирует, поэтому разработчики могут использовать ее в качестве эскиза или чертежа создаваемой системы.

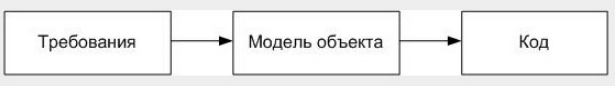
Такой чертеж помогает решить старую проблему. Допустим, команда разработчиков обсудила с пользователями и документировала требования к приложению. Программисты готовы писать код. Один из них (назовем его Боб) берет часть требований, принимает определенные решения и пишет некоторый фрагмент кода. Другой (Джейн) тоже берет часть требований, принимает свои, совершенно отличающиеся от первого, решения по проекту и пишет другой код.

Естественно ожидать различие в стилях программирования; получив одинаковый набор требований, 20 разработчиков создадут 20 различных систем. Таким образом, без подробного обсуждения работы с каждым участником проекта будет трудно понять, какие решения по проекту приняты, из каких элементов состоит система и что представляет собой ее общая структура. Не имея документированного проекта, трудно даже быть уверенным, что созданное приложение — это именно то, чего хотели пользователи.

Традиционно процесс, которому мы следуем, выглядит следующим образом

Традиционный процесс проектирования

Хотя требования и были документированы, весь проект находится в голове



Боба, и никто, кроме Боба, не понимает достаточно хорошо архитектуру системы. Когда Боб оставляет команду, информация уходит вместе с ним. Если вы оказывались в подобной ситуации, то согласитесь, как трудно бывает понять плохо документированную систему.

Модель Rose предлагает совершенно другой подход.

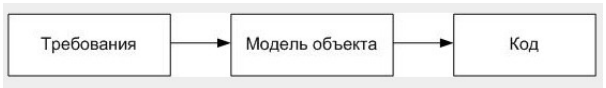
Процесс проектирования в Rose

В этом случае проект уже документирован. Разработчики могут собраться вместе и обсудить принимаемые по проекту решения до фактического написания кода. Вам не нужно больше беспокоиться, что каждый из них пойдет своим путем в проектировании частей одного и того же приложения.

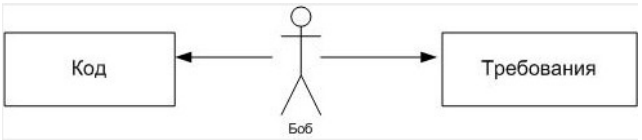
Практическая часть

Создания актеров в Rational Rose.

1. Щелкните правой кнопкой мыши по разделу Use Case View (представление прецедентов) в окне браузера.



2. В появившемся контекстно – зависимом меню выберите команду New => Actor. В список окна браузера будет добавлен новый актер с именем New Class.
3. Выбрав новый пункт списка, введите нужное имя актера.



В модель желательно включить краткое описание каждого актера, в котором нужно указать роль актера при взаимодействии с системой.

Описание актеровв программе Rational Rose осуществляется при выполнении следующих действий:

1. Если окна описания нет на экране, откройте его, выбрав команду меню View => Documentation.
2. Из списка браузера выберите актера, щелкнув по нему мышью.
3. Установите курсор в окно описания и введите текст описания актера.

Создания прецедентов в Rational Rose.

1. Щелкнув пр. клавишей мыши по разделу Use Case View в окне браузера.
2. В появившемся контекстно – зависимом меню выберите команду New => Use Case. В списке браузера появится новый прецедент.
3. Введите для него нужное значение.

В модель следует включить краткое описание прецедентов, которое содержит информацию об их назначении. Такое описание обычно определяется на этапе задумки при выделении прецедентов для системы.

Для добавления краткого описания прецедента:

1. В списке браузера выберите прецедент, щелкнув по нему мышью.
2. Установите курсор в окне описания и наберите краткое описание прецедента. Если окно невидимо, откройте его с помощью команды меню View => Documentation.

Для связи документов, описывающих потоки событий, с прецедентами выполните следующие действия:

1. Щелкните правой кнопкой мыши по прецеденту в окне браузера.
2. В появившемся контекстно – зависимом меню выберите команду Openspecification (открыть параметры)
3. Щелкните на вкладке Files
4. Щелкните правой кнопкой мыши по списку файлов.
5. В появившемся меню выберите команду InsertFile.
6. Укажите нужный файл в стандартном диалоговом окне выбора файла
7. Щелкните по кнопке Open, что бы добавить указанный файл в список.
8. Щелкните по кнопке OK.

Связанные документы добавляются в список браузера.

Создание моделей

Первым шагом в работе с Rose является создание моделей. Их можно строить либо "с нуля", либо взяв за основу существующую каркасную модель. Готовую модель Rose со всеми диаграммами, объектами и другими элементами можно сохранить в одном файле, имеющем расширение . radl (model). Для создания модели:

1. Выберите в меню пункт File\New (Файл\Создать)

Если у вас установлен Мастер каркаса (Framework Wizard), то на экране появится список доступных каркасов (см. рис.). Выберите каркас и щелкните на

кнопке ОК. Если вы не планируете работать с каркасами, щелкните на кнопке Cancel (Отмена)

Сохранение моделей

Как и в случае других приложений, рекомендуется периодически сохранять файлы во время работы с ними. Вся модель сохраняется в одном файле. Кроме того, в отдельном файле можно сохранить журнал.

Для сохранения модели: Выберите в меню пункт File\Save (Файл\ Сохранить) или щелкните мышью на кнопке Save (Сохранить) стандартной панели инструментов. Для сохранения журнала:

1. Выделите окно журнала.
2. Выберите в меню пункт File\Save log As (Файл\Сохранить как).
3. Введите название журнала.
1. Выделите окно журнала.
2. Щелкните мышью на кнопке Save (Сохранить) стандартной панели инструментов.
3. Введите название журнала.

Контрольные вопросы

1. Для чего предназначена среда RationalRose?
2. Объясните необходимость задания цели и точки зрения модели?
3. Перечислите и расскажите назначения кнопок на панели инструментов.
4. Перечислите этапы декомпозиции блока.
5. Расскажите, каким образом на диаграмму добавить блок, дугу.

Лабораторная работа № 3

Тема «Создание контекстной диаграммы IDEF0**»**

Цель работы: ознакомиться с инструментальными средствамисоздания контекстных диаграмм.

Теоретическая часть

Описание системы с помощью IDEF0 называется функциональной моделью. Функциональная модель предназначена для описания существующих бизнес-процессов, в котором используются как естественный, так и графический языки. Для передачи информации о конкретной системе источником графического языка является сама методология IDEF0.

Методология IDEF0 предписывает построение иерархической системы диаграмм - единичных описаний фрагментов системы. Сначала проводится описание системы в целом и ее взаимодействия с окружающим миром (контекстная диаграмма), после чего проводится функциональная декомпозиция - система разбивается на подсистемы и каждая подсистема описывается отдельно (диаграммы декомпозиции). Затем каждая подсистема разбивается на более мелкие и так далее до достижения нужной степени подробности.

Каждая IDEF0-диаграмм а содержит блоки и дуги. Блоки изображают функции моделируемой системы. Дуги связывают блоки вместе и отображают взаимодействия и взаимосвязи между ними.

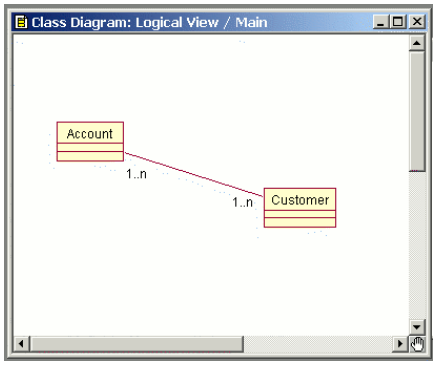
Функциональные блоки (работы) на диаграммах изображаются прямоугольниками, означающими поименованные процессы, функции или задачи, которые происходят в течение определенного времени и имеют

распознаваемые результаты. Имя работы должно быть выражено отглагольным существительным, обозначающим действие.

IDEF0 требует, чтобы в диаграмме было не менее трех и не более шести блоков. Эти ограничения поддерживают сложность диаграмм и модели на уровне, доступном для чтения, понимания и использования.

Программа Rational Roseподдерживает работу с несколькими типами диаграмм UML: диаграммами Вариантов Использования, Последовательности, Кооперативными, Классов, Состояний и Размещения. Для диаграмм каждого типа имеется соответствующая панель инструментов.

В окне диаграммы выводится одна или несколько диаграмм UML вашей модели. При внесении изменений в элементы диаграммы Rose автоматически обновит браузер. Аналогично, при внесении изменений в элемент с помощью браузера Rose автоматически обновит соответствующие диаграммы. Это помогает поддерживать модель в непротиворечивом состоянии.



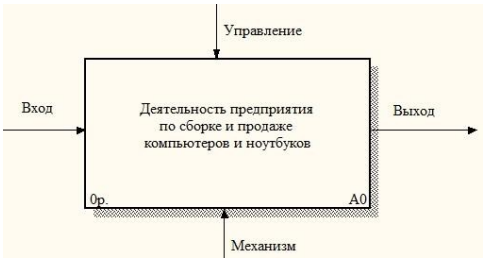
Практическая часть

В данной работе субъектом будет выступать само предприятие, а именно процессы, происходящие внутри него; цель моделирования - воспроизвести бизнес-процессы, происходящие на предприятии (модель AS-IS); точка зрения - с позиции директора как лица, знающего структуру предприятия в целом. После определения контекста моделирования можно приступать к построению контекстной диаграммы (называемой еще "черным ящиком"). Данный тип диаграммы позволяет показать, что подается на вход работы и что является результатом работы, без детализации ее составляющих. Данная диаграмма содержит только одну работу, которая будет представлять всю деятельность предприятия в целом.

Контекстная диаграмма

Любая IDEF0 диаграмма состоит из прямоугольников, называемых работами (activity), и стрелок (arrow). Работа представляет собой некоторую

конкретную функцию в рамках рассматриваемой системы. По требованиям стандарта название каждой работы должно быть выражено отглагольным существительным (например, "Изготовление детали", "Оформление заказа" и т.д.). Каждая из четырех сторон прямоугольника имеет свое определенное значение:



Работа в IDEF0

Вход – это потребляемая или изменяемая работой информация или материал

Выход – информация или материал, которые производятся работой

Управление – процедуры, правила, стратегии или стандарты, которыми руководствуется работа

Механизмы – ресурсы, которые выполняют работу (например, сотрудники, оборудование, устройства)

Для рассматриваемого предприятия входными стрелками будут:

- Заказы клиентов - список компьютеров и их конфигурация, которые клиент желает приобрести
- Комплектующие от поставщиков - комплектующие, полученные от поставщиков, из которых собираются компьютеры и ноутбуки

Выходные стрелки:

- Готовая продукция - собранные компьютеры и ноутбуки
- Заказы поставщикам - список комплектующих, которые предприятие закупает у поставщиков
- Оплата за комплектующие - деньги поставщикам за комплектующие
- Маркетинговые материалы - прайс-листы, рекламки и т.п.

Стрелки управления:

- Законодательство - различные законодательные документы, которыми руководствуется предприятие в процессе своей деятельности
- Правила и процедуры - различные правила и процедуры, которыми руководствуется предприятие в процессе своей деятельности (например, правила сборки и тестирования компьютеров, процедура общения с клиентами и т.п.)

Стрелки механизмов:

- Бухгалтерская система

- Персонал

- Итоговая контекстная диаграмма имеет вид:

Итоговая контекстная диаграмма

Контрольные вопросы

1. Что такое IDEF0?
2. Сколько блоков должно быть в диаграмме в нотации IDEF0?
3. Чем изображаются функциональные блоки (работы) на диаграммах?

Лабораторная работа № 4

Тема «Моделирование бизнес-процессов с помощью инструментальных средств**»**

Цель работы: изучить методологию функционального моделирования IDEF0 и получить практические навыки в моделировании предметной области.

Теоретическая часть

В случае создания ПОдля информационной системы управления предприятием (совокупность средств, методов и персонала для обработки, хранения и выдачи информации) структурный анализ начинается с исследования того, **как организована система** управления предприятием, с обследования **функциональной и информационной структуры** системы управления, чтобы понять, как работает организация, которую собираются автоматизировать. Для описания работы предприятия необходимо построить модель. Такая модель должна быть адекватна предметной области; следовательно, она должна содержать в себе знания всех участников бизнес-процессов организации.

По результатам обследования аналитик строит **модель «как есть»**: обобщенную логическую модель исходной предметной области, отображающую ее функциональную структуру, особенности основной деятельности и информационное пространство, в котором эта деятельность осуществляется. Далее создают **модель «как надо»**: усовершенствованную обобщенную логическую модель, отображающую реорганизованную предметную область или ее часть, которая подлежит автоматизации. Эта стадия анализа содержит элементы проектирования. *Структурные, функциональные* моделисозданные на ранних этапах проектирования программной систем, помогут проектировщику выявить основные функции и составные части проектируемой системы и, по возможности, обнаружить и устранить существенные ошибки. Функциональные диаграммы предметной области помогут понять, как выполняются отдельные операции организации, которые собираются автоматизировать. На этом уровне

определяются все функции, которые выполняет объект, **и процессы**, протекающие в объекте (например, подразделениях предприятия) для выполнения функций, определяются **связи** между функциями, между процессами. **Функция** - преобразование входных потоков в выходные, осуществляемое в соответствии с некоторыми внутренними правилами. Выполнение функции обеспечивает процесс. **Процесс** - совокупность взаимосвязанных **действий (работ)**, преобразующих некоторые входные данные в выходные. Каждый процесс характеризуется определенными задачами и методами их решения, исходными данными, полученными от других процессов, и результатами. Для описания работы организации необходимо построить модель и выделить те процессы, которые должны быть автоматизированы.

Наиболее удобным языком моделирования бизнес-процессов является методология функционального моделирования - IDEF0, предложенный более 20 лет назад Дугласом Россом (SoftTech, Inc.) и называвшийся первоначально SADT - StructuredAnalysisandDesignTechnique.

В IDEF0 система представляется как совокупность взаимодействующих работ или функций. Такая чисто функциональная ориентация является принципиальной - функции системы анализируются независимо от объектов, которыми они оперируют. Это позволяет более четко смоделировать логику и взаимодействие процессов организации.

Под моделью в IDEF0 понимают описание системы (текстовое и графическое), которое должно дать ответ на некоторые заранее определенные вопросы.

Моделируемая система рассматривается как произвольное подмножество Вселенной. Произвольное потому, что, во-первых, мы сами умозрительно определяем, будет ли некий объект компонентом системы, или мы будем его рассматривать как внешнее воздействие, и, во-вторых, оно зависит от точки зрения на систему. Система имеет границу, которая отделяет ее от остальной Вселенной. Взаимодействие системы с окружающим миром описывается как вход (нечто, что перерабатывается системой), выход (результат деятельности системы), управление (стратегии и процедуры, под управлением которых производится работа) и механизм (ресурсы, необходимые для проведения работы). Находясь под управлением, система преобразует входы в выходы, используя механизмы.

Процесс моделирования какой-либо системы в IDEF0 начинается с определения контекста, т.е. наиболее абстрактного уровня описания системы в целом. В контекст входит определение субъекта моделирования, цели и точки зрения на модель.

Под субъектом понимается сама система, при этом необходимо точно установить, что входит в систему, а что лежит за ее пределами; другими словами, мы должны определить, что мы будем в дальнейшем рассматривать как компоненты системы, а что как внешнее воздействие. На определение субъекта системы будет существенно влиять позиция, с которой рассматривается система, и цель моделирования - вопросы, на которые построенная модель должна дать ответ; другими словами, первоначально необходимо определить область (Scope) моделирования. Описание области как системы в целом, так и ее компонентов является основой построения модели. Хотя предполагается, что в течение

моделирования область может корректироваться, она должна быть в основном сформулирована изначально, поскольку именно область определяет направление моделирования и когда должна быть закончена модель. При формулировании области необходимо учитывать два компонента - широту и глубину. Широта подразумевает определение границ модели - мы определяем, что будет рассматриваться внутри системы, а что снаружи. Глубина определяет, на каком уровне детализации модель является завершенной. При определении глубины системы необходимо не забывать об ограничениях времени - трудоемкость построения модели растет в геометрической прогрессии от глубины декомпозиции.

После определения границ модели предполагается, что новые объекты не должны вноситься в моделируемую систему; поскольку все объекты модели взаимосвязаны, внесение нового объекта может быть не просто арифметической добавкой, но в состоянии изменить существующие взаимосвязи. Внесение таких изменений в готовую модель является, как правило, очень трудоемким процессом (так называемая проблема "плавающей области").

Модели **AS-IS и TO-BE.** Целью построения функциональных моделей обычно является выявление наиболее слабых и уязвимых мест деятельности организации, анализ преимуществ новых бизнес-процессов и степени изменения существующей структуры организации бизнеса. Анализ недостатков и "узких мест" начинают с построения модели AS-IS (Как есть), т. е. модели существующей организации работы. Модель AS-IS может строиться на основе изучения документации (должностных инструкций, положений о предприятии, приказов, отчетов и т. п.), анкетирования и опроса служащих предприятия, создания фотографии рабочего дня и другихисточников. Полученная модель AS-IS служит для выявления неуправляемых работ, работ не обеспеченных ресурсами, ненужных и неэффективных работ, дублирующихся работ и других недостатков в организации деятельности предприятия. Исправление недостатков, перенаправление информационных и материальных потоков приводит к созданию модели TO-BE (Как будет) - модели идеальной организации бизнес-процессов. Как правило, строится несколько моделей TO-BE, среди которых определяют наилучший вариант.

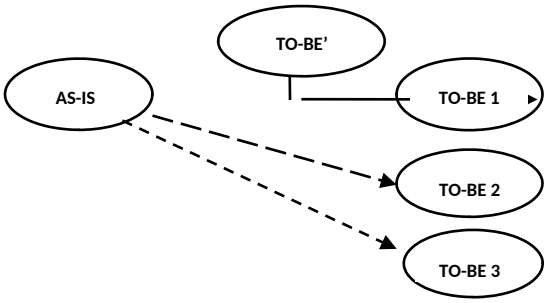


Схема построения моделей "TO-BE" как результат анализа модели "AS-IS"

Технология проектирования программного обеспечения ИС подразумевает сначала создание модели AS-IS, ее анализ и улучшение бизнес-процессов, т. е. создание модели TO-BE, и только на основе модели TO-BE строится модель данных, прототип и затем окончательный вариант ПО.

Практическая часть

1. Построить функциональную диаграмму предметной области, согласно выбранного варианта (Приложение А) с помощью нотации IDEF0.

- Модель должна отражать бизнес-процессы предметной области
- Наличие в модели не менее 3 уровней: контекстная диаграмма и 2 уровня декомпозиции.
- Контекстная диаграмма должна включать не менее 3 функциональных блоков

2. Оформить отчет по лабораторной работе.
3. Представить отчет по лабораторной работе для защиты.

Контрольные вопросы

1. Приведите этапы разработки программного средства.
2. Что представляет собой структурный подход к разработке ПС?
3. В чем преимущество построения модели предметной области при разработке ПС?
4. Для чего строят модели AS-IS и TO-BE?
5. Что такое бизнес-процесс?

Лабораторная работа № 5

Тема «Отображение модели данных в инструментальном средстве**»**

Цель работы: ознакомиться с инструментальными средствами моделирования деловых процессов BPwin.

Теоретическая часть

Начало работы в BPwin

Стандарт IDEF0.

После запуска программы на экране появляется диалоговое окно, в котором предлагается продолжить дальнейшую работу без средства групповой разработки Для продолжения работы необходимо нажать клавишу «Cancel», так как ModelMart –предназначен для коллективной разработки функциональных.

Дальше появится окно «I would like to».в котором в поле Name необходимо указать Имя модели, например, «Изготовление изделия».

Если необходимо создать новую модель, то нужно выполнить File/New, если нужно открыть существующую модель, то File/Open.

Все основные действия с диаграммами, такие как создание, редактирование и т.д, можно выполнить либо с помощью главного меню либо контекстно-зависимого меню (меню, появляющееся при нажатии правой кнопки мыши). Принципы работы с меню являются стандартными для среды Windows: объект сначала делается активным, затем над ним осуществляются необходимые действия.

На основной панели инструментов расположены элементы управления, в основном знакомые по другим Windows-интерфейсам.

Практическая часть

Задание №1. В поле Name: введите имя модели: «Изготовление изделия». Изгруппы Type выберите тип диаграммы Business Process (IDEF0) и нажмите ОК.

При первом открытии программы (при создании новой модели) областьпостроения содержит контекстную диаграмму A-0.

Задание №2. Установите курсор на прямоугольник и введите текст (Изготовлениеизделия) в поле диаграммы, вызвав контекстное меню и выбрав команду Name.

Задание №3. Задайте параметры цветов и их шрифт.

Задание №4. Включите и выключите навигатор модели.

Контрольные вопросы

1. Как начать работу в программе BPwin?
2. Как создать новую модель, открыть существующую модель, указать Имя модели при создании новой модели?
3. Где находится область построения диаграммы?
4. Как включить или выключить навигатор модели, как сохранить модель?
5. Как установить шрифт объекта, как установить цвет объекта?
6. Для чего нужен навигатор модели, какие вкладки имеет навигатор модели?
7. Какую диаграмму содержит область построения диаграммы при создании новой модели?

Лабораторная работа № 6

Тема «Построение диаграммы Вариантов использования и диаграммы Последовательности и генерация кода»

Цель работы: изучить методологию объектно-ориентированного моделирования и получить практические навыки в моделировании спецификаций при разработке программного обеспечения с помощью UML.

Теоретическая часть:

Use Case Diagram – это графическое представление всех или части актеров, прецедентов и их взаимодействия в системе. В каждой системе обычно есть главная диаграмма прецедентов, которая отображает границы системы (актеров) и основное функциональное поведение системы (прецеденты). Другие диаграммы прецедентов могут создаваться при необходимости. Примеры:

- Диаграмма, показывающая все прецеденты для определенного актера
- Диаграмма, показывающая все прецеденты, реализованные на данной итерации
- Диаграмма, показывающая определенный прецедент и все его отношения
- Диаграммы вариантов использования применяются при описании бизнес процессов автоматизируемой предметной области, определении требований к будущей программной системе. Отражает объекты как системы, так и предметной области и задачи, ими выполняемые.

– Не моделируйте связи между действующими лицами.По определению действующие лица находятся вне сферы действия системы. Это означает, что связи между ними также не относятся к её компетенции.

– Не соединяйте сплошной стрелкой (коммуникационной связью) два варианта использования непосредственно. Диаграммы данного типа описывают только, какие варианты использования доступны системе, а не порядок их выполнения. Для отображения порядка выполнения вариантов использования применяют диаграммы деятельности.

– Вариант использования должен быть инициирован действующим лицом. Это означает, что должна быть сплошная стрелка, начинающаяся на действующем лице и заканчивающаяся на варианте использования.

Практическая часть:

1. Построить модель вариантов использования для формирования функциональных требований к разрабатываемому программному обеспечению, согласно выбранного варианта (Приложение А).
2. Оформить отчет по лабораторной работе.
3. Представить отчет по лабораторной работе для защиты.

Порядок построения модели

Чтобы поместить действующее лицо в браузер:

1. Щелкните правой кнопкой мыши на представлении Use Case View в браузере.
2. Выберите в открывшемся меню пункт New далее Actor
3. В браузере появится новое действующее лицо под названием NewClass. Слева от его имени вы увидите пиктограмму действующего лица UML.
4. Выделив новое действующее лицо, введите его имя.

Чтобы поместить вариант использования в браузер:

1. Щелкните правой кнопкой мыши на представлении Use Case View в браузере.
2. Выберите в появившемся меню пункт New далее Use Case
3. Новый вариант использования под названием NewUseCase появится в браузере. Слева от него будет видна пиктограмма варианта использования UML.
4. Выделив новый вариант использования, введите его название.

Построение диаграммы вариантов использования

1. Откройте диаграмму вариантов использования Main.
2. Чтобы поместить действующее лицо или вариант использования на диаграмму, перетащите его мышью из браузера на диаграмму вариантов использования.
3. С помощью кнопок панели инструментов нарисуйте ассоциации между действующими лицами и вариантами использования.
4. Создайте с помощью MS Word текстовые файлы сописаниям варианта использования

Прикрепление файла к варианту использования

1. Щелкните правой кнопкой мыши на варианте использования.
2. В открывшемся меню выберите пункт Open Specification
3. Перейдите на вкладку файлов.

4. Щелкните правой кнопкой мыши на белом поле и из открывшегося меню выберите пункт Insert File.

5. Укажите созданный ранее файл с текстовым описанием и нажмите на кнопку Open, чтобы прикрепить файл к варианту использования.

Контрольные вопросы

1. Как просмотреть диаграмму вариантов использования?
2. Как создать ассоциации между действующими лицами и вариантами использования?
3. Где находится область построения диаграммы?
4. Как добавить файл описания?

Лабораторная работа № 7

Тема «Построение диаграммы Кооперации и диаграммы Развертывания и генерация кода**»**

Цель работы: изучить методологию объектно-ориентированного моделирования и получить практические навыки в моделировании конфигурации системы при разработке программного обеспечения с помощью UML.

Теоретическая часть:

Разработка диаграммы кооперации в среде Rational Rose

Диаграмма кооперации является другим способом визуализации взаимодействия в модели и, как и диаграмма последовательности, оперирует объектами и сообщениями. Особенность работы в среде Rational Rose заключается в том, что этот вид канонической диаграммы создается автоматически после построения диаграммы последовательности и нажатия клавиши <F5>. С помощью этой же клавиши осуществляется переключение между диаграммами последовательности и кооперации.

После того как диаграмма кооперации активизирована, специальная панель инструментов приобретает следующий вид.

Внешний вид специальной панели инструментов для диаграммы кооперации

На этой панели имеются кнопки с пиктограммами объектов и различных типов сообщений. Работа с диаграммой кооперации состоит в добавлении или удалении объектов и сообщений, а также их специфицировании. При этом изменения, вносимые в диаграмму кооперации, автоматически вносятся и в диаграмму последовательности, что можно увидеть, активизировав последнюю нажатием клавиши <F5>.

Разработка логической структуры. После завершения формирования принципов использования системы, наступает этап разработки ее логической структуры. В Rational Rose он именуется "Logical View". Логическое представление, концентрируется на том, как система будет реализовывать поведение, описанное в вариантах использования. Оно дает подробную картину составных частей системы и описывает взаимодействие этих частей. Логическое представление включает, помимо прочего, конкретные требуемые классы,

диаграммы классов и диаграммы состояний. С их помощью конструируется детальный проект создаваемой системы.

Логическое представление содержит:

- классы,
- диаграммы классов. Как правило, для описания системы используется несколько диаграмм классов, каждая из которых отображает некоторое подмножество всех классов системы,
- диаграммы взаимодействия, применяемые для отображения объектов, участвующих в одном потоке событий варианта использования,
- диаграммы состояний,
- пакеты, являющиеся группами взаимосвязанных классов.

Для каждого блока использования строится диаграмма взаимодействия, на которой отображается взаимодействие во времени объектов, выполняющих поставленную задачу. На подобных диаграммах идентифицируются объекты системы и определяются сообщения, с помощью которых эти объекты взаимодействуют

Диаграммы взаимодействия (interaction diagrams) описывают поведение взаимодействующих групп объектов. На такой диаграмме отображается ряд объектов и те сообщения, которыми они обмениваются между собой.

Сообщение (message) – это средство, с помощью которого объект-отправитель запрашивает у объекта получателя выполнение одной из его операций.

Информационное (informative) сообщение – это сообщение, снабжающее объект-получатель некоторой информацией для обновления его состояния.

Сообщение-запрос (interrogative) – это сообщение, запрашивающее выдачу некоторой информации об объекте-получателе.

Императивное (imperative) сообщение – это сообщение, запрашивающее у объекта-получателя выполнение некоторых действий.

Существует два вида диаграмм взаимодействия: диаграммы последовательности (sequence diagrams) и кооперативные диаграммы (collaboration diagrams).

Каждый объект на диаграмме последовательностей сопровождается именем класса, к которому он принадлежит. Конкретный объект является экземпляром некоторого класса. Классы образуют логическую структуру системы.

Результатом данного этапа должна стать главная диаграмма, и детализирующие диаграммы для ее элементов.

На этом этапе следует определить классы, которые необходимы в системе. Экземпляры этих классов уже указаны на диаграммах последовательностей. Классы и их связи отражается в модели в виде диаграммы классов. Группы классов на этих диаграммах могут быть объединены в пакеты.

Проектирование логической структуры следует начинать с определения основных пакетов. Пакет – универсальное средство для группировки элементов модели. Применение пакетов позволяет сделать модель более обозримой. Пакеты могут быть вложенными друг в друга. Классы, составляющие каждый пакет, детализируются на вложенной диаграмме.

Идентификация основных абстракций заключается в предварительном определении набора классов системы (классов анализа) на основе описания предметной области.

Все классы и диаграммы, описывающие системный проект, помещаются в пакет с именем DesignModel.

Диаграммы классов, реализующие варианты использования и диаграммы взаимодействия, отражающие взаимодействие объектов в процессе реализации сценариев варианта использования, помещаются в кооперацию с именем данного варианта использования и стереотипом «use-case realization». Все кооперации помещаются в пакет с именем Use-CaseRealization. Связь между вариантом использования и его реализацией изображается на специальной диаграмме трассировки.

Практическая часть:

1. Для реализации сценариев варианта использования постройте
a. диаграммы классов, реализующих вариант использования,
b. диаграммы взаимодействия, отражающие взаимодействие объектов в процессе.
c. кооперативные диаграммы для построенных диаграмм взаимодействия
2. Все построенные диаграммы помещаются в кооперацию с именем данного варианта использования и стереотипом «use-case realization». Все кооперации помещаются в пакет с именем Use Case Realizations.
3. Оформить отчет по лабораторной работе.
4. Представить отчет по лабораторной работе для защиты.

Порядок построения модели

Создание классов

1. Щелкните правой кнопкой мыши на представлении Logical View.
2. Выберите в открывшемся меню пункт New\Class. Новый класс под названием NewClass появится в браузере.
3. Выделите его и введите имя класса.
4. Щелкните правой кнопкой мыши на созданном классе.
5. В открывшемся меню выберите пункт Open Specification.
6. В поле стереотипа выберите необходимый стереотип (Boundary, Control, Entity) и нажмите на кнопку ОК.
7. Откройте диаграмму Main и перетащите созданные классы.

Создание пакетов и диаграммы Traceabilities:

8. Щелкните правой кнопкой мыши на представлении Logic View.
9. В открывшемся меню выберите пункт New\Package.
10. Создайте пакет Use-Case Realizations, затем внутри него – пакеты соответствующие построенным вариантам использования.
11. В каждом из пакетов создайте соответствующие кооперации (каждая кооперация представляет собой вариант использования со стереотипом «use-case realization», который задается в спецификации варианта использования).
12. Создайте в пакете Use-Case Realizations новую диаграмму вариантов использования с названием Traceabilities, которая показывает связь между вариантом использования и его реализацией (диаграмма трассировки).

Создание диаграмм взаимодействия

Настройка

1. В меню модели выберите пункт Tools далее Options.
2. Перейдите на вкладку диаграмм.
3. Контрольные переключатели Sequence Numbering, Collaboration Numbering должны быть помечены, а Focus of Control – нет.
4. Нажмите ОК, чтобы выйти из окна параметров.

Создание диаграммы последовательности

1. Щелкните правой кнопкой мыши на кооперации
2. В открывшемся меню выберите пункт New далее Sequence Diagram.
3. Назовите новую диаграмму.
4. Дважды щелкните на ней, чтобы открыть ее.

Добавление на диаграмму действующего лица, объектов и сообщений

1. Перетащите действующее лицо из браузера на диаграмму.
2. Перетащите классы из браузера на диаграмму.
3. На панели инструментов нажмите кнопку Object Message (Сообщение объекта).
4. Проведите мышью от линии жизни действующего лица к линии жизни объекта
5. Выделив сообщение, введите его имя.
6. Повторите действия 3 – 5, чтобы поместить на диаграмму остальные сообщения (для рефлексивного сообщения используется кнопка Message to Self).

Соотнесение сообщений с операциями

1. Щелкните правой кнопкой на тексте сообщении
2. В открывшемся меню выберите пункт <new operation>. Появится окно спецификации операции.
3. В поле имени оставьте имя сообщения.
4. Нажмите на кнопку ОК, чтобы закрыть окно спецификации операции и вернуться на диаграмму.
5. Повторите действия 1 – 4, пока не соотнесете с операциями все остальные сообщения.

Создание примечаний

Чтобы поместить на диаграмму примечание:

1. Нажмите на панели инструментов кнопку Note.
2. Щелкните мышью в том месте диаграммы, куда собираетесь поместить примечание.
3. Выделив новое примечание, введите туда текст.
4. Чтобы прикрепить примечание к элементу диаграммы, на панели инструментов нажмите кнопку Anchor Notes To Item (Прикрепить примечания к элементу).
5. Нажав левую кнопку мыши, проведите указатель от примечания до элемента диаграммы, с которым оно будет связано. Между примечанием и элементом возникнет штриховая линия.
6. Чтобы создать примечание-ссылку на другую диаграмму создайте пустое примечание (без текста) и перетащите на него из браузера нужную диаграмму.

Чтобы поместить на диаграмму текстовую область:

1. На панели управления нажмите кнопку Text Box.
2. Щелкните мышью внутри диаграммы, чтобы поместить туда текстовую область.
3. Выделив эту область, введите в нее текст.

Создание кооперативной диаграммы

Для создания кооперативной диаграммы достаточно открыть диаграмму последовательности и нажать клавишу F5.

Контрольные вопросы

1. Что такоедиаграмма кооперации?
2. Как переключиться с диаграммы последовательности на диаграмму кооперации?
3. Когда наступает этап разработки логической структуры системы?

Лабораторная работа № 8

Тема «Построение диаграммы Деятельности, диаграммы Состояний и диаграммы Классов и генерация кода**»**

Цель работы: получить навыки построения диаграмм Деятельности, Состояний, Классов.

Теоретическая часть:

Приемы работы с диаграммами состояний в ArgoUML

Диаграммы состояний используются для моделирования динамических аспектов системы. По большей части под этим подразумевается моделирование поведения реактивных объектов. Реактивным называется объект, поведение которого лучше всего характеризуется его реакцией на события, произошедшие вне его собственного контекста. У реактивного объекта есть четко выраженный жизненный цикл, когда текущее поведение обусловлено прошлым. Диаграммы состояний можно присоединять к классам, прецедентам или системе в целом для визуализации, специфицирования, конструирования и документирования динамики отдельного объекта.

Диаграмма состояний (StateChart, StateMachine diagram) показывает автомат, фокусируя внимание на потоке управления от состояния к состоянию.

Автомат (State machine) - это описание последовательности состояний, через которые проходит объект на протяжении своего жизненного цикла, реагируя на события, - в том числе описание реакций на эти события.

Состояние (State) - это ситуация в жизни объекта, на протяжении которой он удовлетворяет некоторому условию, осуществляет определенную деятельность или ожидает какого-то события.

Событие (Event) - это спецификация существенного факта, который происходит во времени и пространстве. В контексте автоматов событие - это стимул, способный вызвать срабатывание перехода.

Переход (Transition) - это отношение между двумя состояниями, показывающее, что объект, находящийся в первом состоянии, должен выполнить некоторые действия и перейти во второе состояние, как только произойдет определенное событие и будут выполнены заданные условия.

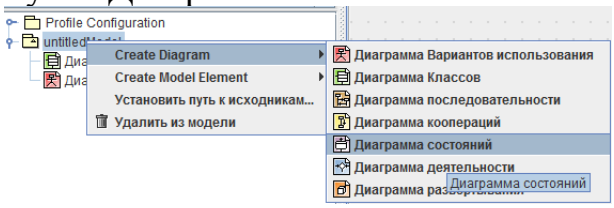
Действие (Action) - это атомарное вычисление, которое приводит к смене состояния или возврату значения.

Диаграмма состояний изображается в виде графа с вершинами и ребрами

Практическая часть

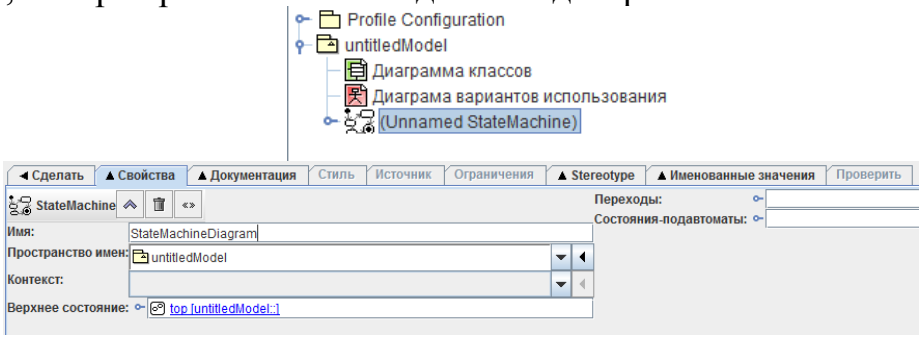
Для того чтобы создать новую диаграмму состояний в ArgoUML, необходимо кликнуть по кнопке  панели инструментов, либо в окне браузера

проекта кликнуть правой кнопкой по необходимой модели, раскрыть меню CreateDiagram и выбрать пункт Диаграмма состояний.



Создание диаграммы состояний.

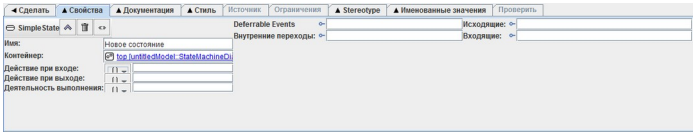
Далее следует выделить только что созданную диаграмму и задать ей имя в окне свойств, которое расположено под окном диаграммы.



Свойства диаграммы состояний.

Добавление нового состояния

Для создания нового состояния нужно щелкнуть по кнопке *Простое состояние* на панели инструментов и затем по свободному месту диаграммы. При выделении любого элемента с помощью инструмента  *Select* (при создании элемента он выделяется автоматически) под окном диаграммы открывается окно редактирования свойств элемента.

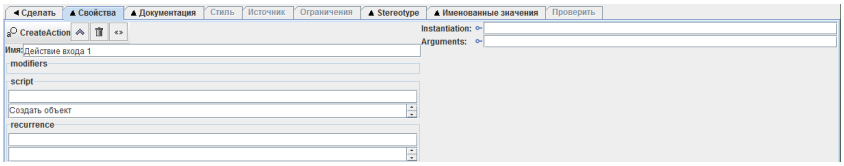


Свойства простого состояния.

Для каждого состояния можно определять действия (как в автомате Мура). В ArgoUML можно задать 3 действия для каждого состояния: действие при входе в состояние, действие при выходе из состояния и деятельность выполнения. Для того чтобы задать действие, нужно раскрыть выпадающее меню и выбрать тип действия. Всего в ArgoUML определены 8 типов действий:

- действие вызова;
- действие создания;
- действие уничтожения;
- действие возврата;
- действие отправки;
- действие конечного состояния;
- не интерпретируемое действие;
- последовательность действий.

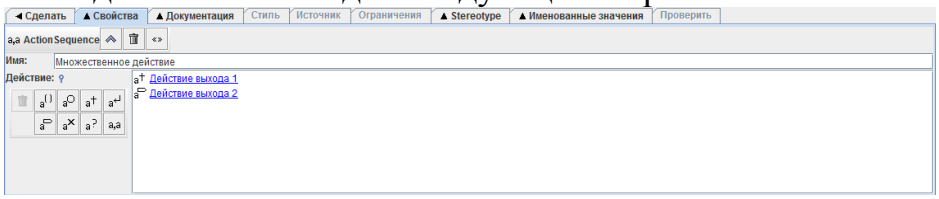
Для выбора типа действия следует кликнуть по одному из пунктов. В результате откроется окно редактирования действия:



Свойства действия входа (тип: создание объекта).

В этом окне можно задать имя действия и, собственно, содержание действия (группа script). Именно содержание многострочного поля script (в примере – «Создать объект») отобразится на диаграмме. В однострочном поле этой группы (первом по порядку) можно указать язык скрипта.

В качестве действия можно выбрать последовательность. Редактирование последовательности действий выглядит следующим образом:



Свойства деятельности выполнения (тип: последовательность действий)

С помощью меню слева, аналогичного меню создания действий, можно добавлять действия к последовательности. Для редактирования отдельных действий нужно кликнуть по произвольному действию 2 раза. Редактирование каждого отдельного действия аналогично редактированию единичных действий.

Псевдосостояния

Псевдосостояния – начальное и конечное состояния объекта или процесса. Каждая диаграмма состояний должна начинаться начальным состоянием и заканчиваться конечным. Для добавления на диаграмму начального и конечного состояний используются кнопки  и  соответственно.

Рассмотрим следующий пример. Составим диаграмму состояний для класса Order (Заказ), поскольку в нашей модели он наиболее часто будет менять свое состояние. Заказ может находиться в нескольких состояниях:

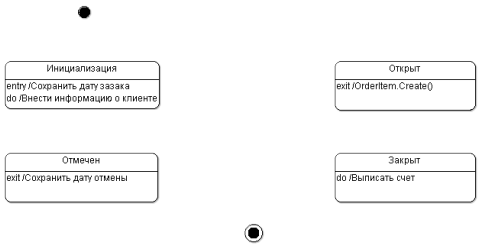
- при создании заказа он переходит в состояние *Инициализация,* в котором выполняются некоторые предварительные действия;
- после завершения инициализации заказ переходит в состояние *Открыт*, в котором к заказу добавляются новые пункты. Выход из этого состояния возможен или в случае отмены заказа, или в случае заполнения всех необходимых пунктов заказа;
- если заполнены все необходимые пункты заказа, то он переходит в состояние *Закрыт*, в котором происходит выписка счета. Выход из этого состояния произойдет только после того, как счет будет выписан;
- если заказ отменен, то из состояния *Открыт* он переходит в состояние *Отменен*. При выходе из этого состояния происходит удаление всех пунктов заказа.

Разместим все состояния объекта на диаграмме, в том числе начальное и конечное. Определим некоторые действия, которые выполняются в данных состояниях.

- на входе в состояние Инициализация – действие «Сохранить дату заказа»;

- деятельность состояния Инициализация – действие «Внести информацию о клиенте»;
- на выходе из состояния Открыт – действие создания пункта заказа;
- на выходе из состояния Отменен – действие «Сохранить дату отмены»;
- деятельность состояния Закрыт – действие «Выписать счет».

С помощью инструмента Broom можно выровнять элементы относительно друг друга.

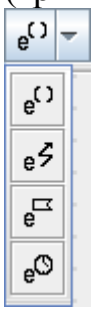


Состояния объекта Entry.

Добавление переходов

Переходы добавляются на диаграмму с помощью инструмента New Transition. После добавления перехода или при его выделении, под областью диаграммы открывается окно редактирования перехода.

Наиболее важными свойствами связей являются Переключающее событие (триггер), Результат (действие) и Сторожевое условие (выражение).



Триггер – некоторое событие, которое возбуждает изменение состояния объекта. Для того чтобы задать переключающее событие, нужно раскрыть выпадающее меню и выбрать его тип. Всего в ArgoUML определены 4 типа событий:

- событие вызова;
- событие изменения;
- событие сигнала;
- событие времени.

Вернемся к примеру. Добавим переходы между состояниями. Необходимо создать следующие триггеры:

- Заказ создан (событие сигнала) – переход из начального состояния в состояние «Инициализация»;
- Заказ отменен (событие изменения) – переход из состояния «Открыт» в состояние «Отменен»;
- Счет выписан (событие изменения) – переход из состояния «Закрыт» в конечное состояние.

Также определим несколько сторожевых условий:

- Инициализация завершена – условие перехода из состояния «Инициализация» в состояние «Открыт»;

- Заполнены не все пункты заказа – условие перехода из состояния «Открыт» в себя же;
- Заполнены все позиции заказа – условие перехода из состояния «Открыт» в состояние «Закрыт».

Необходимо задать недостающие действия для переходов:

- Добавление пункта заказа – действие создания при переходе из состояния «Открыт» в себя;
- Удаление записи (OrderItem.Delete()) – действие уничтожения при переходе из состояния «Отменен» в конечное состояние.

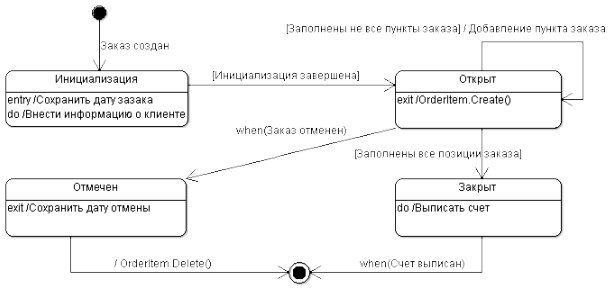


Диаграмма состояний объекта Entry.

Контрольные вопросы

1. Приведите определения следующих понятий: автомат, состояние, событие, действие, переход.
2. Для чего предназначена диаграмма состояний?
3. К каким элементам системы может быть привязана диаграмма состояний?
4. Как изображается диаграмма состояний?
5. Какие существуют типы состояний?
6. Для чего служат действия, события, сторожевые условия?

Лабораторная работа № 9

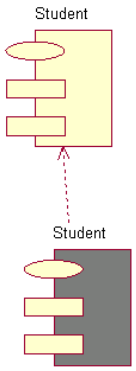
Тема «Построение диаграммы компонентов и генерация кода**»**

Цель работы: изучить методологию объектно-ориентированного моделирования и получить практические навыки в генерации программы на основе построенных моделей.

Теоретическая часть

Диаграммы компонентов показывают, как выглядит модель на физическом уровне. На них изображены компоненты программного обеспечения и связи между ними. При этом на такой диаграмме выделяют два типа компонентов: исполняемые компоненты и библиотеки кода.

Каждый класс модели (или подсистема) преобразуется в компонент исходного кода. После создания они сразу добавляются к диаграмме компонентов. Между отдельными компонентами изображают зависимости, соответствующие зависимостям на этапе компиляции или выполнения программы. Пример диаграммы компонентов показан на рисунке



Представление компонентов содержит:

– Компоненты, являющиеся физическими модулями кода.

– Диаграммы компонентов.

– Пакеты, являющиеся группами связанных компонентов.

Практическая часть:

1. Для реализации построенной системы
a. постройте диаграмму компонентов
b. выполните проверку корректности модели
c. выполните генерацию кода,
2. Оформить отчет по лабораторной работе.
3. Представить отчет по лабораторной работе для защиты.

Порядок построения модели

Создание диаграммы компонентов:

1. Дважды щелкните мышью по главной диаграмме компонентов в представлении компонентов.
2. На панели инструментов нажмите кнопку Package Specification.
3. Поместите спецификацию пакета на диаграмму.
4. Введите имя спецификации пакета и укажите в окне спецификации язык программирования для генерации кода.
5. На панели инструментов нажмите кнопку Package Body.
6. Поместите тело пакета на диаграмму.
7. Введите имя тела пакета и укажите в окне спецификации язык генерации кода
8. На панели инструментов нажмите кнопку Dependency.
9. Проведите линию зависимости от тела пакета к спецификации пакета.

Соотнесение классов с компонентами:

1. В логическом представлении браузера найдите необходимый для генерации класс.
2. Перетащите этот класс на спецификацию пакета компонента в представлении компонентов браузера. В результате класс будет соотнесен со спецификацией пакета компонента.

Процесс генерации кода состоит из четырех основных шагов:

1. Проверка корректности модели.
2. Установка свойств генерации кода.
3. Выбор класса, компонента или пакета.
4. Генерация кода.

Для проверки модели:

1. Выберитевменю Tools\Check Model.
2. Проанализируйте все найденные ошибки в окне журнала, используя команду View\Log.

К наиболее распространенным ошибкам относятся такие, например, как сообщения на диаграмме последовательности или кооперативной диаграмме, не соотнесенные с операцией, либо объекты этих диаграмм, не соотнесенные с классом.

С помощью пункта меню Check Model можно выявить большую часть неточностей и ошибок в модели. Пункт меню Access Violations позволяет обнаруживать нарушения правил доступа, возникающие тогда, когда существует связь между двумя классами разных пакетов, но связи между самими пакетами нет.

Для того чтобы обнаружить нарушение правил доступа:

1. Выберитевменю Report\Show Access Violations.
2. Проанализируйте все нарушения правил доступа в окне.

Можно установить несколько параметров генерации кода для классов, атрибутов, компонентов и других элементов модели. Этими свойствами определяется способ генерации программ. Для каждого языка в Rose предусмотрен ряд определенных свойств генерации кода. Перед генерацией кода рекомендуется анализировать эти свойства и вносить необходимые изменения.

Для анализа свойств генерации кода

1. выберите Tools\Options, а затем вкладку соответствующего языка.
2. в окне списка можно выбрать класс, атрибут, операцию и другие элементы модели.

Для каждого языка в этом списке указаны свои собственные элементы модели. При выборе разных значений на экране появляются разные наборы свойств.

Для изменения свойства генерации кода для одного класса, атрибута, одной операции и т.д. нужно

1. открыть окно спецификации элемента модели. Выбрать вкладку языка (C++, Java,...) и изменить свойства.Все изменения, вносимые в окне спецификации элемента модели, оказывают влияние только на этот элемент.

При генерации кода за один раз можно создать класс, компонент или целый пакет. Код генерируется с помощью диаграммы или браузера. При генерации кода из пакета можно выбрать или пакет логического представления на диаграмме классов, или пакет представления компонентов на диаграмме компонентов. При выборе пакета логического представления генерируются все классы этого пакета. При выборе пакета представления компонентов генерируются все компоненты этого пакета.

После выбора класса или компонента на диаграмме выберите в меню соответствующий вариант генерации кода. Сообщения об ошибках, возникающих в процессе генерации кода, будут появляться в окне журнала.

Во время генерации кода Rose выбирает информацию из логического и компонентного представлений модели и генерирует большой объем «скелетного» (skeletal) кода:

Генерация кода

1. Откройте диаграмму компонентов системы.
2. Выберите все объекты на диаграмме компонентов.
3. Выберите Tools\(язык для генерации кода)\Code Generation в меню.
4. Выполните генерацию кода.
5. Просмотрите результаты генерации (меню Tools\\(язык для генерации кода)\ Browse Header и Tools\\(язык для генерации кода)\Browse Body.

Контрольные вопросы

1. Что такое диаграмма компонентов, что она показывает?
2. Как выполнить генерацию кода?
3. Как выполнить проверку корректности построенной модели?

Лабораторная работа № 10

Тема «Построение диаграмм потоков данных и генерация кода**»**

Цель работы: ознакомиться с инструментальными средствами моделирования деловых процессов, их классификацией и примерами.

Теоретическая часть:

RAMUS Educational - Программа, предназначенная для построения визуальных диаграмм, помогающих лучше понять различные процессы на предприятиях. Поддерживает методологии IDEF0 и DFD.

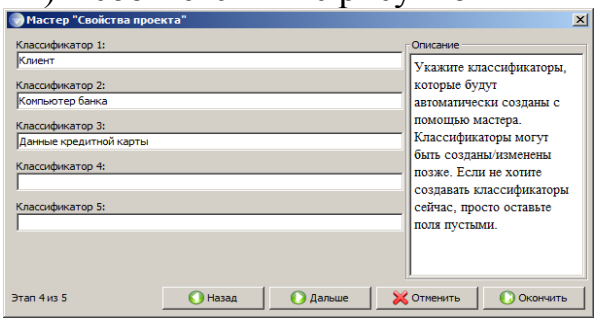
Ramus - это программа, при помощи которой можно создавать визуальные диаграммы, используемые для наглядного отображения различных бизнес процессов. Данное решение будет крайне полезно на "мозговых штурмах" и собрания сотрудников предприятия. Помимо визуализации разных процессов и задач, создаваемые программой диаграммы также неплохо подходят для классификации и систематизации различных данных. Главное преимущество Ramus Educational заключается в том, что она поддерживает сразу две популярных методологии: DFD и IDEF0.

Практическая часть

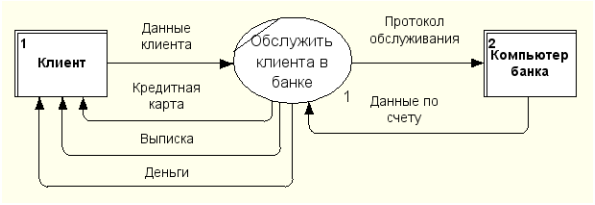
Программное обеспечение: RAMUSEducational.

Порядок выполнения

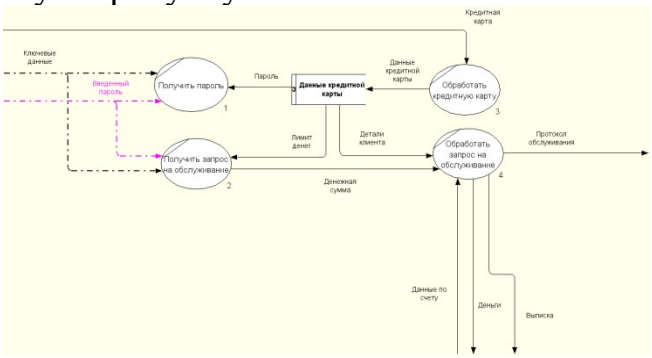
1. Создайте новый проект. При редактировании свойств проекта укажите нотацию DFD.
2. Введите названия классификаторов (элементы DFD: внешняя сущность и накопитель данных) в соответствии с рисунком



3. С помощью панели элементов для диаграммы DFD создайте диаграмму, как показано на рисунке ниже.



4. Выполните декомпозицию диаграммы DFD на 4 процесса.
5. Создайте диаграмму по рисунку ниже.



6. По ниже приведенным вариантам создайте диаграмму DFD.

Варианты заданий DFD

Вариант 1.

Создать диаграмму потоков данных процесса «ОБСЛУЖИТЬ» при работе библиотекаря с клиентами, начиная работу с заказа клиентом нужного ему издания из хранилища.

Вариант 2.

Создать диаграмму потоков данных процесса «ПРОВЕСТИ МАРКЕТИНГОВЫЕ ИССЛЕДОВАНИЯ», подробно рассмотрев все процессы, происходящие при этом. В качестве внешних сущностей можно выбрать «КЛИЕНТ» и «РЫНОК».

Вариант 3.

Создать диаграмму потоков данных процесса «ОБЕСПЕЧИТЬ ПРЕДПРЯТИЕ ТОВАРОМ» при работе отдела сбыта с поставщиками.

Вариант 4.

Создать диаграмму потоков данных процесса "ПЛАНИРОВАТЬ ДЕЯТЕЛЬНОСТЬ ПРЕДПРИЯТИЯ», учитывая финансовую, хозяйственную и прочие деятельности предприятия.

Вариант 5.

Создать диаграмму потоков данных процесса «СОЗДАТЬ ПРОГРАММУ» при работе программиста над разработкой и созданием ПО.

Вариант 6.

Создать диаграмму потоков данных процесса «РАЗРАБОТАТЬ КОНСАЛТИНГОВЫЙ ПРОЕКТ», учитывая основные этапы при проведении консалтинга:

1. анализ первичных требований;
2. проведение обследования деятельности предприятия;
3. построение моделей «как есть» и «как должно быть»;
4. оценка эффективности деятельности предприятия;
5. реорганизация деятельности;
6. разработка системного проекта;

7. разработка предложений по автоматизации;

8. выбор, разработка и внедрение новой информационной системы. Создать словарь данных, описав все хранилища данных и внешниесущности.

Вариант 7.

Создать диаграмму потоков данных процесса «ОБЕСПЕЧИТЬ ПРОДАЖУ ТОВАРА» при работе отдела сбыта крупного предприятия, работающего как на местном рынке, так и на мировом.

Вариант 8.

Создать диаграмму потоков данных процесса «РАБОТА ТУРИСТИЧЕСКОЙ ФИРМЫ» при работе проектировщика с использованием компьютера.

Вариант 9.

Создать диаграмму потоков данных процесса «ОБСЛУЖИТЬ» при покупке акций на рынке ценных бумаг.

Контрольные вопросы

1. Для чего предназначена программа Ramus?
2. Какие методологии поддерживает программа Ramus?
3. Как выполнить проверку корректности построенной модели?

Лабораторная работа № 11

Тема «Обоснование выбора технических средств**»**

Цель работы:

1. Закрепление имеющихся знаний о средствах разработкипрограммного обеспечения информационных систем.

2. Приобретение навыков работы в современных интегрированных средах разработки программного обеспечения.

3. Приобретение навыков разработки клиентского программного обеспечения ИС с применением принципов методологии *RAD*.

Теоретическая часть

Быстрая разработка приложений *RAD*(*RapidApplicationDevelopment*)является одной из современных методологий разработки программного обеспечения. Как и другие методологии(*MSF*, *RUP* и др.) *RAD* описывает итеративный подход к организации процесса разработки ПО и соответствующую модель жизненного цикла. Методологию *RAD* также часто связывают с технологией *визуального программирования* и применением современных *интегрированных сред разработки* программного обеспечения.

Методология *RAD* основывается на визуализации процессасоздания программного кода приложений и поддерживается инструментальным ПО, которое предоставляет разработчикам средства *визуального программирования.* Применение средств визуального программирования позволяет значительного ускоритьпроцесс разработки приложений, а также уменьшить трудоёмкость работы по модификации уже готовой программы, внесениюв неё необходимых дополнений или изменений.

Средства быстрой разработки приложений, как правило, основываются на объектно-ориентированной компонентной архитектуре. Процедура разработки интерфейса средствами *RAD* сводится к набору последовательных операций, включающих:

1) размещение компонентов интерфейса в нужном месте;

2) задание моментов времени их появления на экране;

3) настройку связанных с ними атрибутов и событий.

Интегрированная среда разработки (ИСР) является средством, с помощью которого выполняются проектирование, программирование, тестирование и отладка прикладных программ.

Примерами современных ИСР, поддерживающих методологию*RAD* и технологию визуального программирования, являются *MicrosoftVisualStudio*, *EmbarcaderoRADStudio*, *IntelliJIDEA*, *MonoDevelop*и другие

Практическая часть

Вариант индивидуального задания определяет информационную систему, для которой необходимо разработать клиентскоепрограммное обеспечение.

В процессе выполнения лабораторной работы необходимо:

1. Выполнить анализ требований к информационной системе. Составить перечень функциональных требований кклиентскому приложению. Сформулировать общие требования к пользовательскому интерфейсу.

2. Разработать проект пользовательского интерфейса приложения. С помощью интегрированной среды разработкисоздать макеты экранных форм с размещёнными на нихэлементами интерфейса.

3. Разработать прототип клиентского приложения, пользуясь средствами визуального программирования интегрированной среды разработки.

4. Реализовать необходимый функционал приложения добавлением программного кода для обработки системныхсобытий и действий пользователя.

5. Выполнить тестирование общей работоспособности и отдельных функциональных возможностей разработанногоприложения. Исправить возможные ошибки.

6. Выполнить верификацию функциональных возможностейразработанного приложения, сравнивая их с имеющимсяперечнем функциональных требований.

7. Разработать документ «Руководство пользователя» с описанием назначения и функциональных возможностей
клиентского приложения создаваемой системы.

Варианты индивидуальных заданий

1. ИС «Телефонный справочник» (поисковая система).
2. ИС «Библиотека» (информационно-справочная система, поисковая система).
3. ИС «Издательство» (СЭДО, САБП).
4. ИС «Поликлиника» (СЭДО, информационно-справочная система).
5. ИС «Школа» (обучающая система, информационно-справочная система).
6. ИС «Ателье» (САБП).
7. ИС «Склад» (САБП).
8. ИС «Торговля» (САБП, СЭДО).
9. ИС «Автосалон» (САБП, СЭДО).
10. ИС «Продажа подержанных автомобилей» (информаци-

онно-справочная система, поисковая система).
11. ИС «Автосервис» (САБП).
12. ИС «Пассажирское автопредприятие» (САБП, СЭДО).
13. ИС «Диспетчерская служба такси» (ГИС, СЭДО).
14. ИС «Агентство по продаже авиабилетов» (информационно-справочная система, поисковая система).
15. ИС «Туристическое агентство» (информационно-справочная система, поисковая система).
16. ИС «Гостиница» (информационно-справочная система, СЭДО).__

Контрольные вопросы

1. Средства разработки программного обеспечения ИС.
2. Программные платформы, технологии программирования и инструментальные средства разработки.
3. Интегрированные среды разработки.
4. Современные средства разработки ПО.
5. Методология быстрой разработки приложений *RAD*.

Лабораторная работа № 12

Тема «Стоимостная оценка проекта**»**

Цель работы: ознакомиться с инструментальными средствами моделирования деловых процессов, их классификацией и примерами.

Теоретическая часть

Стоимостный анализ (ABC) и свойства, определяемые пользователем (UDP)

обычно сначала строится функциональная модель существующей организации работы - AS-IS (Как есть). После построения модели AS-IS проводится анализ бизнес-процессов, потоки данных и объектов перенаправляются и улучшаются, в результате строится модель TO-BE. Как правило, строится несколько моделей TO-BE, из которых по какому-либо критерию выбирается наилучшая. Проблема состоит в том, что таких критериев много и непросто определить важнейший. Для того чтобы определить качество созданной модели с точки зрения эффективности бизнес-процессов, необходима система метрики, т. е. качество следует оценивать количественно.

BPwin предоставляет аналитику два инструмента для оценки модели - стоимостный анализ, основанный на работах (Activity Based Costing, ABC), и свойства, определяемые пользователем (User Defined Properties, UDP). ABC является широко распространенной методикой, используемой международными корпорациями и государственными организациями (в том числе Департаментом обороны США) для идентификации истинных движителей затрат в организации.

Стоимостный анализ представляет собой соглашение об учете, используемое для сбора затрат, связанных с работами, с целью определить общую стоимость процесса. Стоимостный анализ основан на модели работ, потому что количественная оценка невозможна без детального понимания функциональности

предприятия. Обычно ABC применяется для того, чтобы понять происхождение выходных затрат и облегчить выбор нужной модели работ при реорганизации деятельности предприятия (Business Process Re-engineering, BPR). С помощью стоимостного анализа можно решить такие задачи, как определение действительной стоимости производства продукта, определение действительной стоимости поддержки клиента, идентификация работ, которые стоят больше всего (те, которые должны быть улучшены в первую очередь), обеспечение менеджеров финансовой мерой предлагаемых изменений, и др.

ABC может проводиться только тогда, когда модель работы последовательная (следует синтаксическим правилам IDEF0), корректная (отражает бизнес), полная (охватывает всю рассматриваемую область) и стабильная (проходит цикл экспертизы без изменений), другими словами, создание модели работы закончено.

Практическая часть

ABC включает следующие основные понятия:

- объект затрат - причина, по которой работа выполняется, обычно, основной выход работы, стоимость работ есть суммарная стоимость объектов затрат ("Готовое изделие").

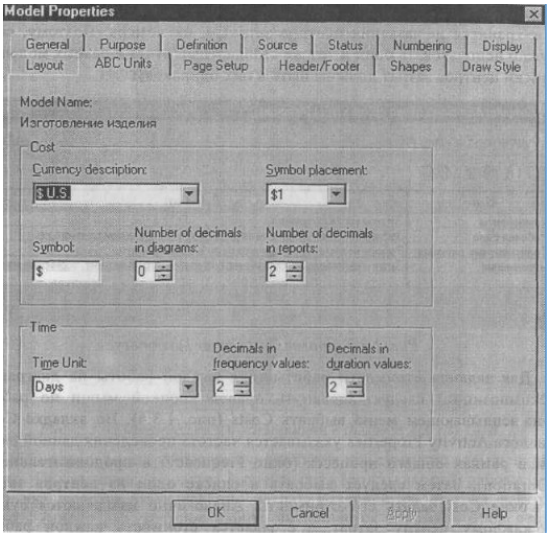
движитель затрат - характеристики входов и управлений работы ("Сырье", "Чертеж"), которые влияют на то, как выполняется и как долго длится работа;

центры затрат, которые можно трактовать как статьи расхода.

Иллюстрация терминов ABC

При проведении стоимостного анализа в BPwin сначала задаются единицы измерения времени и денег. Для задания единиц измерения следует вызвать диалог Model Properties (меню Edit/Model Properties), вкладка ABC Units.

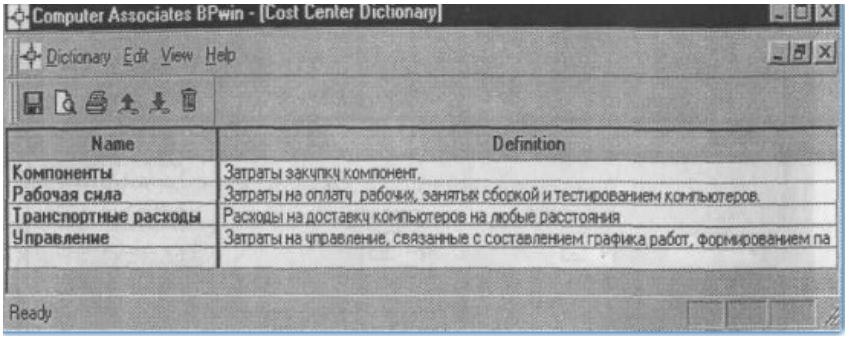
Если в списке выбора отсутствует необходимая валюта (например, рубль), ее можно добавить. Символ валюты по умолчанию берется из настроек Windows. Диапазон измерения времени в списке Unit of measurment достаточен для большинства случаев - от секунд до лет.



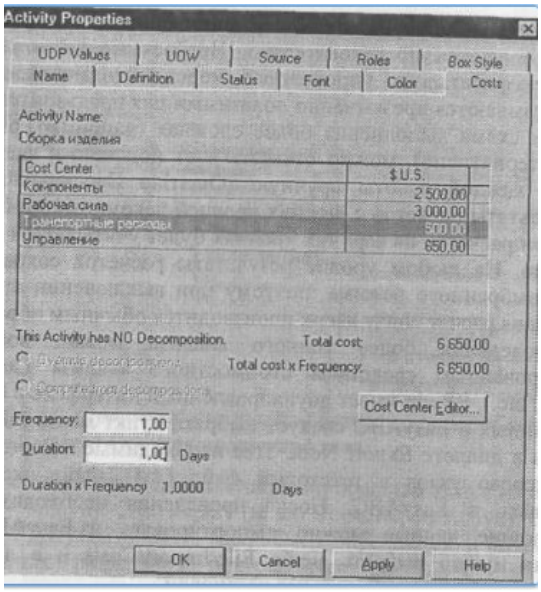
Настройка единиц измерения валюты и времени

Затем описываются центры затрат (cost centers). Для внесения центров затрат необходимо вызвать диалог Cost Center Dictionary (меню Dictionary /Cost Center.

Каждому центру затрат следует дать подробное описание в окне Definition. Список центров затрат упорядочен. Порядок в списке можно менять при помощи стрелок, расположенных справа от списка. Задание определенной последовательности центров затрат в списке, во-первых, облегчает последующую работу при присвоении стоимости работам, а во-вторых, имеет значение при использовании единых стандартных отчетов в разных моделях. Хотя BPwin сохраняет информацию о стандартном отчете в файле BPWINRPT.INI, информация о центрах затрат и UDP сохраняется в виде указателей, т. е. хранятся не названия центров затрат, а их номера. Поэтому, если нужно использовать один и тот же стандартный отчет в разных моделях, списки центров затрат должны быть в них одинаковы.

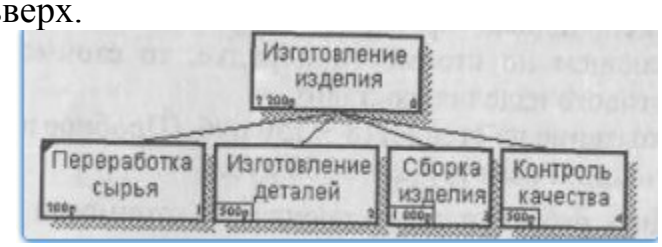


Для задания стоимости работы (для каждой работы на диаграмме декомпозиции) следует щелкнуть правой кнопкой мыши по работе и на всплывающем меню выбрать Costs (рис. 1.3.4). Во вкладке Costs диалога Activity Properties указывается частота проведения данной работы в рамках общего процесса (окно Frequency) и продолжительность (Duration). Затем следует выбрать в списке один из центров затрат и в окне Cost задать его стоимость. Аналогично назначаются суммы по каждому центру затрат, т. е. задается стоимость каждой работы по каждой статье расхода. Если в процессе назначения стоимости возникает необходимость внесения дополнительных центров затрат, диалог Cost Center Editor вызывается прямо из диалога Activity Cost соответствующей кнопкой.



Задание стоимости работ в диалоге Activity Cost

Общие затраты по работе рассчитываются как сумма по всем центрам затрат. При вычислении затрат вышестоящей (родительской) работы сна чала вычисляется произведение затрат дочерней работы на частоту работы (число раз, которое работа выполняется в рамках проведения родительской работы), затем результаты складываются. Если во всех работах модели включен режим Compute from Decompositions, подобные вычисления автоматически проводятся по всей иерархии работ снизу вверх.



Вычисление затрат родительской работы

Этот достаточно упрощенный принцип подсчета справедлив, если работы выполняются последовательно. Встроенные возможности BPwin позволяют разрабатывать упрощенные модели стоимости, которые тем не менее оказываются чрезвычайно полезными для предварительной оценки затрат. Если схема выполнения более сложная (например, работы производятся альтернативно), можно отказаться от подсчета и задать итоговые суммы для каждой работы вручную (Override Decompositions). В этом случае результаты расчетов с нижних уровней декомпозиции будут игнорироваться, при расчетах на верхних уровнях будет учитываться сумма, заданная вручную. На любом уровне результаты расчетов сохраняются независимо от выбранного режима, поэтому при выключении опции Override Decompositions расчет снизу вверх производится обычным образом.

Для проведения более тонкого анализа можно воспользоваться специализированным средством стоимостного анализа EasyABC (ABC Technology, Inc.). BPwin имеет двунаправленный интерфейс с EasyABC. Для экспорта данных в EasyABC следует выбрать пункт меню File/Export/Node Tree, задать в диалоге Export Node Tree необходимые настройки и экспортировать дерево узлов в текстовый файл (.txt). Файл экспорта можно импортировать в EasyABC. После проведения необходимых расчетов результирующие данные

можно импортировать из EasyABC в BPwin. Для импорта нужно выбрать меню File/Import/Costs и в диалоге Import Activity Costs выбрать необходимые установки.

Результаты стоимостного анализа могут существенно повлиять на очередность выполнения работ. Рассмотрим пример, изображенный на рис. 1.3.6. Предположим, что для оценки качества изделия необходимо провести три работы:

внешний осмотр-стоимость 50 руб.;

пробное включение - стоимость 150 руб.;

испытание на стенде - стоимость 300 руб.

Предположим также, что с точки зрения технологии очередность проведения работ несущественна, а вероятность выявления брака одинакова (50 %). Пусть необходимо проверить восемь изделий. Если проводить работы в убывающем по стоимости порядке, то стоимость, затраченная на получение готового изделия, составит:

300 руб. (Испытание на стенде) (8+150 руб. (Пробное включение) (4 +

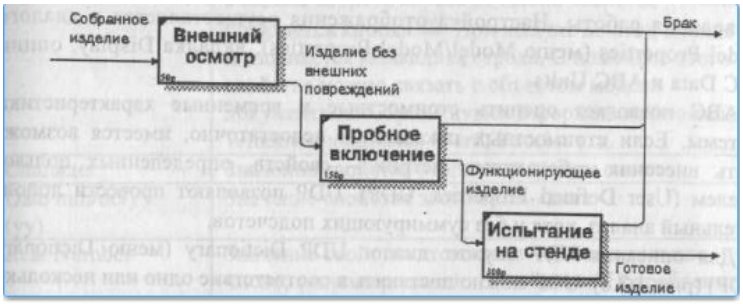
+ 50 руб. (Внешний осмотр) (2 = 3100 руб.

Если проводить работы в возрастающем по стоимости порядке, то стоимость, затраченная на получение готового изделия, составит:

50 руб. (Внешний осмотр) (8+150 руб. (Пробное включение) (4 +

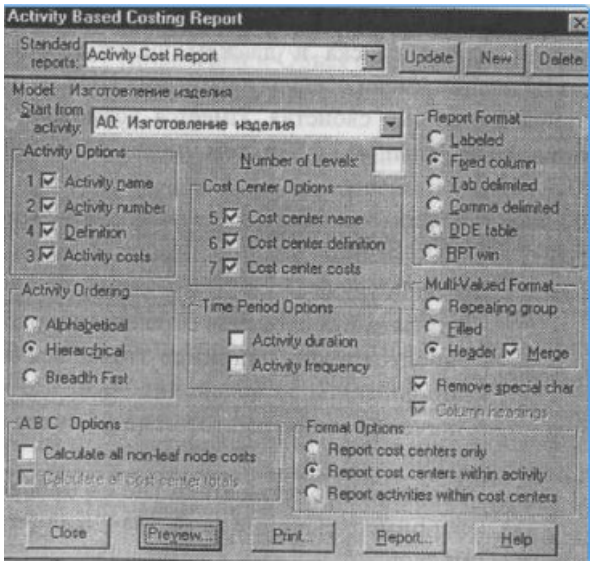
+ 300 руб. (Испытание на стенде) (2 = 1600 руб.

Следовательно, с целью минимизации затрат первой должна быть выполнена наиболее дешевая работа, затем - средняя по стоимости и в конце - наиболее дорогая.



Фрагмент диаграммы декомпозиции работы "Контроль качества "

Результаты стоимостного анализа наглядно представляются на специальном отчете BPwin - Activity Cost Report (меню Tools/Report/Activity Cost Report). Отчет позволяет документировать имя, номер, определение и стоимость работ, как суммарную, так и раздельно по центрам затрат (рис. 1.3.7).



Диалог настройки отчета по стоимости работ

Результаты отображаются и непосредственно на диаграммах. В левом нижнем углу прямоугольника работы может показываться либо стоимость (по умолчанию), либо продолжительность, либо частота проведения работы. Настройка отображения осуществляется в диалоге Model Properties (меню Model/Model Properties), вкладка Display, опции ABC Data и ABC Units.

ABC позволяет оценить стоимостные и временные характеристики системы. Если стоимостных показателей недостаточно, имеется возможность внесения собственных метрик - свойств, определенных пользователем (User Defined Properties, UDP). UDP позволяют провести дополнительный анализ, хотя и без суммирующих подсчетов.

Для описания UDP служит диалог UDP Dictionary (меню Dictionary /UDP) (рис. 1.3.8). UDP можно поставить в соответствие одно или несколько ключевых слов. Ключевые слова могут быть использованы для отбора UDP при печати отчетов или при присвоении свойств работам и стрелкам. Ключевые слова должны быть описаны в словаре UDP Keyword List (рис. 1.3.9). Для внесения нового ключевого слова следует щелкнуть по кнопке Ш. и в таблице диалога UDP Keyword List задать значение ключевого слова.

Для создания нового свойства (UDP) следует в словаре UDP Dictionary перейти к нижней строке списка и дважды щелкнуть по полю Name. В режиме редактирования имени следует внести имя UDP. В поле UDP Type (рис. 1.3.10) описывается тип свойства. Имеется возможность задания 18 различных типов UDP (табл. 1.3.1), в том числе управляющих команд и массивов.

Таблица Типы UDP и их использование

Тип	*Использование*
Text	При задании свойства стрелки или работы просто вносится текст, например это может быть просто дополнительное пояснение
Paragraph Text	Значение свойства этого типа – текст в несколько строк
Integer	Значение свойства этого типа – целое число, например значение свойства “Количество баллов”

ТипИспользование

Character ListМассив символов. Значения свойства этого типа

(Single selection) должны быть определены в диалоге UDP Dictionary (поле Value). Объекту модели можно присваивать только одно значение из предварительно заданного списка

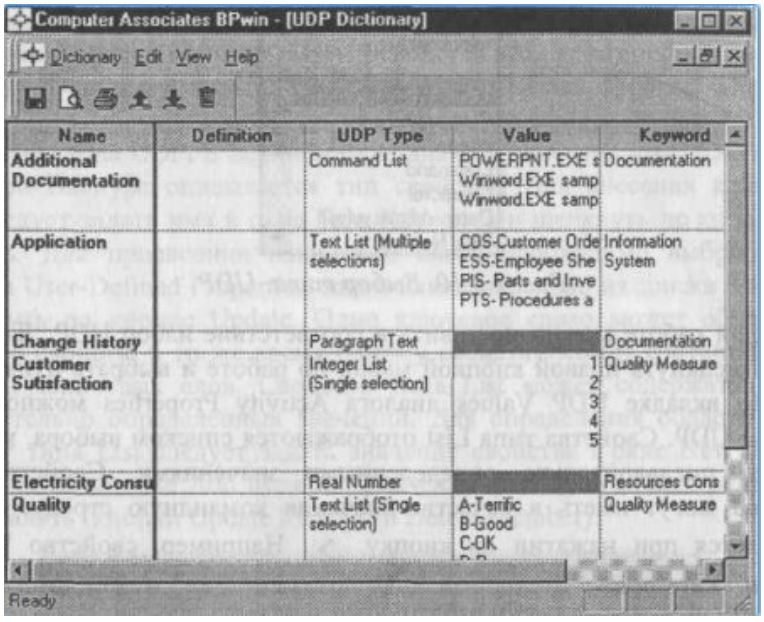
Text List (Multiple Массив строк (множественный выбор).Значения selections)свойства этого типа должны быть определены в диалоге UDP Dictionary (поле Value). Объекту модели можно присваивать одновременно несколько значений из предварительно заданного

Списка Integer ListМассив целых чисел (множественный выбор).

(MultipleЗначения свойства этого типа должны быть selections)определены в диалоге UDP Dictionary (поле Value).

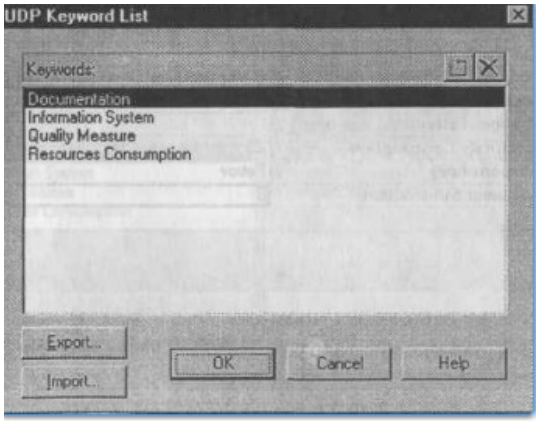
Объекту модели можно присваивать одновременно несколько значений из предварительно заданного списка Date List (Multiple Массив дат (множественный выбор).Значения selections)свойства этого типа должны быть определены в диалоге UDP Dictionary (поле Value).Объекту модели можно присваивать одновременно несколько значений из предварительно заданного списка Real Number List Массив действительных чисел (множественный (Multiplевыбор).Значения свойства этого типа должны быть selections)определены в диалоге UDP Dictionary (поле Value).

Объекту модели можно присваивать одновременно несколько значений из предварительно заданного списка Character ListМассив символов (множественный выбор). (MultipleЗначения свойства этого типа должны быть selections)определены в диалоге UDP Dictionary (поле Value). Объекту модели можно присваивать одновременно несколько значений из предварительно заданного списка

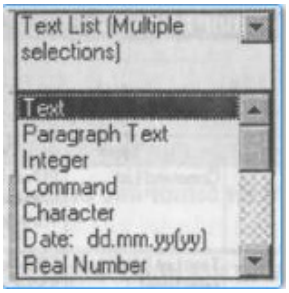


Диалог описания UDP

Для присвоения свойству ключевого слова следует перейти к полю Keyword и выбрать из списка необходимые ключевые слова. Одному свойству может соответствовать несколько разных ключевых слов, одно ключевое слово может соответствовать разным свойствам.

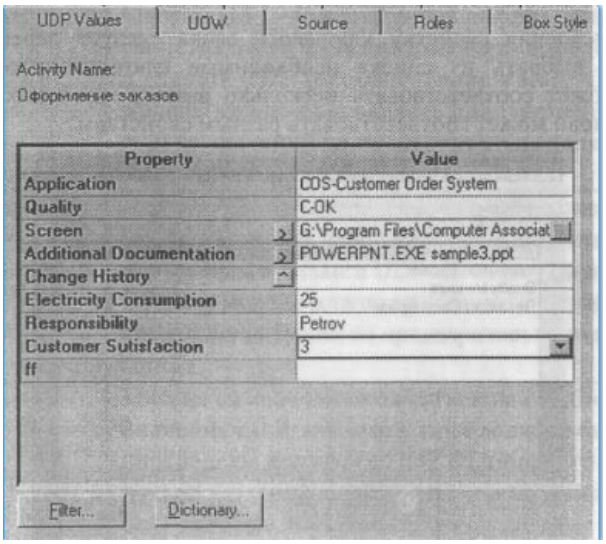


Диалог описания ключевых слов UDP



Выбор типа UDP

Каждой работе можно поставить в соответствие набор UDP. Для этого следует щелкнуть правой кнопкой мыши по работе и выбрать пункт меню UDP. Во вкладке UDP Values диалога Activity Properties можно задать значения UDP. Свойства типа List отображаются списком выбора, который заполнен предварительно определенными значениями. Свойства типа Command могут иметь в качестве значения командную строку, которая выполняется при нажатии на кнопку > . Например, свойство "Спецификации" категории "Дополнительная документация" может иметь значение C:MSOffice97OfficeWINWORD.EXE specl.doc.

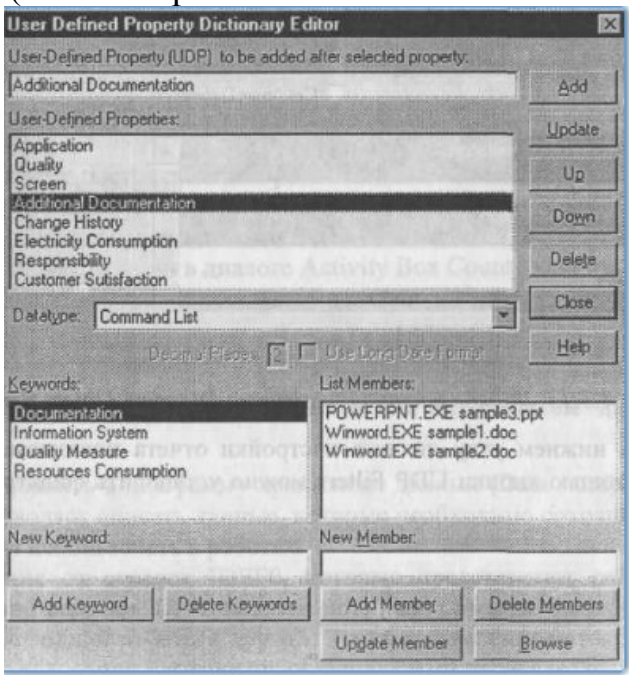


Задание значений UDP

Кнопка Filter служит для задания фильтра по ключевым словам UDP. По умолчанию в списке показываются свойства всех категорий.

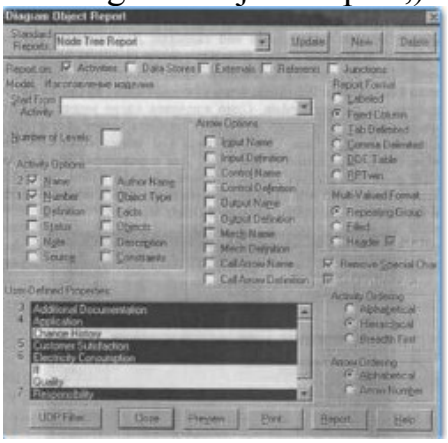
Кнопка Dictionary вызывает диалог User Defined Property Dictionary Соис. 1-3-12), который позволяет создавать и редактировать как UDP, так ключевые слова UDP. В верхнем окне диалога вносится имя UDP, в списке выбора Datatype

описывается тип свойства. Для внесения ключевого слова следует задать имя в окне New Keywords и щелкнуть по кнопке Add Keywords. Для присвоения ключевого слова необходимо выбрать UDP из списка User-Defined Properties, затем ключевое слово из списка Keywords и щелкнуть по кнопке Update. Одно ключевое слово может объединять несколько свойств, в то же время одному свойству может соответствовать несколько ключевых слов. Свойство типа List может содержать массив предварительно определенных значений. Для определения области значений UDP типа List следует задать значение свойства в окне New Member и щелкнуть по кнопке Add Member. Значения из списка можно редактировать и удалять (кнопки Update Member и Delete Member).



Диалог User Defined Property Dictionary

Результат задания значений UDP можно проанализировать в отчете Diagram Object Report (меню Tools/Report/Diagram Object Report,).



Диалог настройки отчета Diagram Object Report

Контрольные вопросы

1. Какие инструменты для оценки моделей представляет BPWin?
2. Опишите методику ABC.
3. Что такое стоимостной анализ?.

Лабораторная работа № 13

Тема «Построение и обоснование модели проекта**»**

Цель работы:

1. Закрепление имеющихся знаний о моделях жизненного цикла ИС и современных методологиях разработки программного обеспечения.
2. Приобретение навыков анализа требований, условий и ограничений проекта создания ИС и оценки трудоёмкости его реализации.
3. Приобретение навыков составления планов разработки ИС на основе разных моделей жизненного цикла.

Теоретическая часть

Жизненный цикл (ЖЦ) информационной системы – непрерывный процесс, который начинается с момента принятия решения о необходимости создания системы и заканчивается в моментеё полного изъятия из эксплуатации.

Модель жизненного цикла ИС – структура, описывающаяпроцессы, действия и задачи, которые осуществляются в ходеразработки, функционирования и сопровождения программногообеспечения в течение всей жизни ИС, от определения требований до завершения её использования.

К настоящему времени наибольшее распространение получили следующие основные модели ЖЦ:

1) каскадная (водопадная) модель и её варианты;
2) инкрементная модель;
3) спиральная модель.

Каскадная или *водопадная* модель ЖЦ является классической моделью однократного прохода, которая описывает линейную последовательность этапов создания ИС.

Спиральная модель ЖЦ относится к эволюционным моделям. Каждый виток раскручивающейся спирали соответствуетразработке одной (начальной, промежуточной или окончательной) версии ИС и представляет собой полный цикл разработки,начиная с анализа и заканчивая внедрением.

Прототип – версия ИС, предназначенная для демонстрациизаказчику некоторых ключевых свойств будущего продукта. Создание прототипа позволяет вовлечь заказчика в разработку информационной системы в самом начале работы.

Практическая часть:

Вариант индивидуального задания определяет информационную систему, для создания которой необходимо составитьплан разработки на основе каскадной и спиральной моделей жизненного цикла.

В процессе выполнения лабораторной работы необходимо:

1. Подготовить исходные данные. Исходными данными дляпланирования являются:

1.1. Общее описание некоторой ИС (назначение, областьприменения, решаемые задачи, технологические особенности реализации и внедрения).

1.2. Ограничения и условия разработки (требования заказчика, возможности команды разработчиков, срокиразработки, бюджет проекта и т.д.).

2. Составить план разработки ИС с применением каскадного подхода:

2.1. Составить эскизный план разработки ИС на основекаскадной модели ЖЦ.

2.2. Для этапа «Анализ требований» составить документ«Техническое задание» с подробным описаниемфункциональных требований к ИС.
2.3. Для этапа «Проектирование» составить документ«Технический проект» с описанием проектных решений (архитектура системы, логическая структура базы данных, решения по реализации пользовательского интерфейса и т.д.).
2.4. Для этапа «Тестирование» составить документ «Плантестирования» с описанием методики тестирования иконтрольных тестов.
2.5. Для этапа «Внедрение» составить документ «Планввода ИС в эксплуатацию».
2.6. Уточнить параметры календарного плана разработкиИС, учитывая ограничения и условия разработки.
2.7. Объединить календарный план разработки и составленные документы в единый отчёт «Разработка ИСна основе каскадной модели ЖЦ».
3. Составить план разработки ИС с применением итеративного подхода:
3.1. Разделить весь процесс создания и внедрения ИС нанесколько итераций.
3.2. На основе имеющихся документов (см. пункты 2.2 –2.5) для каждой итерации составить отдельный комплект документов.
3.3. Составить календарный план итеративной разработки ИС.
3.4. Объединить план итеративной разработки и составленные документы в единый отчёт «Разработка ИСна основе спиральной модели ЖЦ».

Варианты индивидуальных заданий

1. ИС «Телефонный справочник» (поисковая система).
2. ИС «Библиотека» (информационно-справочная система,поисковая система).
3. ИС «Издательство» (СЭДО, САБП).
4. ИС «Поликлиника» (СЭДО, информационно-справочнаясистема).
5. ИС «Школа» (обучающая система, информационно-справочная система).
6. ИС «Ателье» (САБП).
7. ИС «Склад» (САБП).
8. ИС «Торговля» (САБП, СЭДО).
9. ИС «Автосалон» (САБП, СЭДО).
10. ИС «Продажа подержанных автомобилей» (информационно-справочная система, поисковая система).
11. ИС «Автосервис» (САБП).
12. ИС «Пассажирское автопредприятие» (САБП, СЭДО).
13. ИС «Диспетчерская служба такси» (ГИС, СЭДО).
14. ИС «Агентство по продаже авиабилетов» (информационно-справочная система, поисковая система).
15. ИС «Туристическое агентство» (информационно-справочная система, поисковая система).
16. ИС «Гостиница» (информационно-справочная система, СЭДО).

Контрольные вопросы

1. Современные методологии разработки информационных систем.
2. Жизненный цикл информационных систем.
3. Этапы жизненного цикла: анализ, проектирование, программирование, тестирование, эксплуатация.
4. Модели жизненного цикла.
5. Методология Microsoft Solutions Framework.

6. Методология Rational Unified Process.

Лабораторная работа № 14

Тема «Установка и настройка системы контроля версий с разграничением ролей**»**

Цель работы: ознакомиться с инструментальными средствами моделирования деловых процессов, их классификацией и примерами.

Теоретическая часть

Быстрая разработка приложений RAD (Rapid Application Development)является одной из современных методологий разработки программного обеспечения. Как и другие методологии(MSF, RUP и др.) RAD описывает итеративный подход к организации процесса разработки ПО и соответствующую модель жизненного цикла. Методологию RAD также часто связывают с технологией визуального программирования и применением современных интегрированных сред разработки программного обеспечения.

Методология RAD основывается на визуализации процессасоздания программного кода приложений и поддерживается инструментальным ПО, которое предоставляет разработчикам средства визуального программирования. Применение средств визуального программирования позволяет значительного ускоритьпроцесс разработки приложений, а также уменьшить трудоёмкость работы по модификации уже готовой программы, внесениюв неё необходимых дополнений или изменений.

Средства быстрой разработки приложений, как правило, основываются на объектно-ориентированной компонентной архитектуре. Процедура разработки интерфейса средствами RAD сводится к набору последовательных операций, включающих:

1) размещение компонентов интерфейса в нужном месте;

2) задание моментов времени их появления на экране;

3) настройку связанных с ними атрибутов и событий.

Интегрированная среда разработки (ИСР) является средством, с помощью которого выполняются проектирование, программирование, тестирование и отладка прикладных программ.

Примерами современных ИСР, поддерживающих методологиюRAD и технологию визуального программирования, являются MicrosoftVisual Studio, Embarcadero RAD Studio, IntelliJ IDEA, MonoDevelop.

Практическая часть

Вариант индивидуального задания определяет информационную систему, для которой необходимо разработать клиентскоепрограммное обеспечение.

В процессе выполнения лабораторной работы необходимо:

1. Выполнить анализ требований к информационной системе. Составить перечень функциональных требований кклиентскому приложению. Сформулировать общие требования к пользовательскому интерфейсу.

2. Разработать проект пользовательского интерфейса приложения. С помощью интегрированной среды разработкисоздать макеты экранных форм с размещёнными на нихэлементами интерфейса.

3. Разработать прототип клиентского приложения, пользуясь средствами визуального программирования интегрированной среды разработки.

4. Реализовать необходимый функционал приложения добавлением программного кода для обработки системныхсобытий и действий пользователя.

5. Выполнить тестирование общей работоспособности и отдельных функциональных возможностей разработанногоприложения. Исправить возможные ошибки.

6. Выполнить верификацию функциональных возможностейразработанного приложения, сравнивая их с имеющимсяперечнем функциональных требований.

7. Разработать документ «Руководство пользователя» с описанием назначения и функциональных возможностейклиентского приложения создаваемой системы.

Варианты индивидуальных заданий

1. ИС «Телефонный справочник» (поисковая система).
2. ИС «Библиотека» (информационно-справочная система,поисковая система).
3. ИС «Издательство» (СЭДО, САБП).
4. ИС «Поликлиника» (СЭДО, информационно-справочнаясистема).
5. ИС «Школа» (обучающая система, информационно-справочная система).
6. ИС «Ателье» (САБП).
7. ИС «Склад» (САБП).
8. ИС «Торговля» (САБП, СЭДО).
9. ИС «Автосалон» (САБП, СЭДО).
10. ИС «Продажа подержанных автомобилей» (информационно-справочная система, поисковая система).
11. ИС «Автосервис» (САБП).
12. ИС «Пассажирское автопредприятие» (САБП, СЭДО).
13. ИС «Диспетчерская служба такси» (ГИС, СЭДО).
14. ИС «Агентство по продаже авиабилетов» (информационно-справочная система, поисковая система).
15. ИС «Туристическое агентство» (информационно-справочная система, поисковая система).
16. ИС «Гостиница» (информационно-справочная система, СЭДО).

Контрольные вопросы

1. Средства разработки программного обеспечения ИС.
2. Программные платформы, технологии программирования и инструментальные средства разработки.
3. Интегрированные среды разработки.
4. Современные средства разработки ПО.
5. Методология быстрой разработки приложений RAD.

Лабораторная работа № 15

Тема «Проектирование и разработка интерфейса пользователя**»**

Цель работы: ознакомиться со способами разработки интерфейсов.

Теоретическая часть:

Важную роль в системе ООП играют интерфейсы. Они определяют некоторый абстрактный функционал, не имеющий конкретной реализации, который уже реализуют классы, наследующие эти интерфейсы. Определение интерфейса похоже на определения класса: интерфейс также может содержать свойства, методы и события. Чтобы объявить интерфейс, надо использовать ключевое слово ***Interface*** (обратите внимание, что имена интерфейсов обычно начинаются с заглавной буквы **I**).

Мы разрабатываем программу для банка. В нашей программе мы можем определить три класса: Person, который описывает человека, Employee, который описывает сотрудника банка, и класс Client, который будет представлять клиента банка. Очевидно, что классы Employee и Client будут производными от класса Person. И так как все объекты будут представлять либо сотрудника банка, либо клиента, то напрямую мы от класса Person создавать объекты не будем. Поэтому имеет смысл сделать его абстрактным.

Практическая часть:

Создайте новое консольное приложение, добавьте класс.

```
Public MustInherit Class Person

    Public Property FirstName() As String
    Public Property LastName() As String
    'Абстрактный метод
    Public MustOverride Sub Display()

    Public Sub New(fName As String, lName As String)
        FirstName = fName
        LastName = lName
    End Sub

End Class

Public Class Employee
    Inherits Person

    Public Property Bank As String

    Public Overrides Sub Display()
        Console.WriteLine(FirstName & " " & LastName & " works in " & Bank)
    End Sub

    Public Sub New(fName As String, lName As String, _bank As String)
        MyBase.New(fName, lName)
        Bank = _bank
    End Sub

End Class

Public Class Client
    Inherits Person

    Public Property Bank As String

    Public Overrides Sub Display()
        Console.WriteLine(FirstName & " " & LastName & " has an account in bank " & Bank)
    End Sub

    Public Sub New(fName As String, lName As String, _bank As String)
        MyBase.New(fName, lName)
        Bank = _bank
    End Sub

End Class
```

Добавим в приложение интерфейс IAccount, который будет содержать методы и свойства, которые понадобятся при работе с счетом клиента. Чтобы

добавить интерфейс, в меню Project выберите пункт Add New Item... и в появившемся списке выберите пункт Code File (**Добавить – Пользовательский элемент управления - Интерфейс**). Код интерфейса:

```
Public Interface IAccount
    'Текущая сумма на счете
    ReadOnly Property CurentSum() As Integer
    'Метод для добавления денег на счет
    Sub Put(sum As Integer)
    'Метод для снятия денег со счета
    Sub Withdraw(sum As Integer)
    'Процент начислений
    ReadOnly Property Procentage() As Integer
End Interface
```

Обратите внимание, что методы и свойства не имеют реализации, в этом они сближаются с абстрактными методами абстрактных классов. Сущность этого интерфейса проста: он определяет два свойства для текущей суммы денег на счете и ставки процента по вкладам и два метода для добавления денег на счет и изъятия денег. Теперь надо реализовать интерфейс в классе Client, так как клиент у нас обладает счетом. Чтобы реализовать интерфейс, нам надо использовать ключевое слово **Implements**. Изменим класс Client следующим образом:

```
Public Class Client
    Inherits Person
    Implements IAccount

    'Переменная для хранения суммы
    Dim _sum As Integer
    'Переменная для хранения процента
    Dim _procentage As Integer

    Public Property Bank As String

    'Текущая сумма на счете
    ReadOnly Property CurentSum() As Integer Implements IAccount.CurentSum
        Get
            Return _sum
        End Get
    End Property
    'Метод для добавления денег на счет
    Sub Put(sum As Integer) Implements IAccount.Put
        _sum += sum
    End Sub
    'Метод для снятия денег со счета
    Sub Withdraw(sum As Integer) Implements IAccount.Withdraw
        If sum <= CurentSum Then
            _sum -= sum
        End If
    End Sub
    'Процент начислений
    ReadOnly Property Procentage() As Integer Implements IAccount.Procentage
        Get
            Return _procentage
        End Get
    End Property

    Public Overrides Sub Display()
        Console.WriteLine(FirstName & " " & LastName & " has an account in bank " & Bank)
    End Sub

    Public Sub New(fName As String, lName As String, _bank As String, _sum As Integer)
        MyBase.New(fName, lName)
        Bank = _bank
        Me._sum = _sum
    End Sub
```

Обратите внимание, что класс, реализующий интерфейс, обязан реализовать все его свойства, методы и события.

Зачем же нужны интерфейсы, если по сути они ничего не делают, только объявляют методы и свойства? Во-первых, интерфейсы позволяют реализовать концепцию множественного наследования. Если некоторый класс может иметь только один базовый класс, то при этом он может реализовать множество интерфейсов. Во-вторых, они более гибки по сравнению с классами, так как не содержат конкретной реализации.

Придумайте реализацию созданного интерфейса.

Контрольные вопросы

1. На чем основан объектный подход к проектированию?
2. Что такое объект?
3. Что такое класс?
4. Перечислите основные свойства объектного подхода к проектированию?

Лабораторная работа № 16

Тема «Разработка графического интерфейса пользователя**»**

Цель работы: ознакомиться со способами разработки графических интерфейсов.

Теоретическая часть

Важную роль в системе ООП играют интерфейсы. Они определяют некоторый абстрактный функционал, не имеющий конкретной реализации, который уже реализуют классы, наследующие эти интерфейсы. Определение интерфейса похоже на определения класса: интерфейс также может содержать свойства, методы и события. Чтобы объявить интерфейс, надо использовать ключевое слово ***Interface*** (обратите внимание, что имена интерфейсов обычно начинаются с заглавной буквы **I**).

Мы разрабатываем программу для банка. В нашей программе мы можем определить три класса: Person, который описывает человека, Employee, который описывает сотрудника банка, и класс Client, который будет представлять клиента банка. Очевидно, что классы Employee и Client будут производными от класса Person. И так как все объекты будут представлять либо сотрудника банка, либо клиента, то напрямую мы от класса Person создавать объекты не будем. Поэтому имеет смысл сделать его абстрактным.

Практическая часть

Создайте в приложении MSACCESS базу данных, состоящую из двух таблиц: «Физические лица» и «Сотрудники».

Таблица «Физические лица» содержит следующие поля:

- «Код» – используется в качестве первичного ключа, тип Integer (Счетчик), индексированное поле;
- «Фамилия», «Имя», «Отчество», «Телефон», «Индекс», «Страна», «Город», «Адрес» – текстовые поля;
- «Дата рождения» – типа Дата/Время;

- «Пол» – поле типа Текстовый.

Таблица «Сотрудники» содержит следующие поля:

- «Код» – используется в качестве первичного ключа, тип Integer, индексированное поле;
- «Должность» – текстовое поле;
- «Разряд», «Зарплата», «Рейтинг» – числовые поля;

Сохраните полученный результат в файле *firma.mdb*или*firma.accdb*.

Разработаем форму для просмотра и редактирования информации, содержащейся в таблице «Физические лица». Научимся использовать управляющие элементы Open File и Save File для просмотра файлов.

Последовательность действий для создания простых форм.

1.Запуститеплатформу Microsoft Visual Studio.

Диалоговое окно Open File

2. Для создания нового приложения выполните команду Файл – Создать – Проект (**File⇒NewProject)**. Выберите **Приложение WindowsForms**. Назовите проект **Manipulating Files**. Измените название формы, которая выбрана по умолчанию, на fclsManipulatingFiles, текст в форме — на Manipulating Files. Добавьте в форму новое текстовое поле и настройте его свойства, как показано в данной таблице.

Свойство	*Значение*
Name	txtSource
Location	95,8
Size	184,20
Text	*(оставить пустым)*

3. Добавьте в проект диалоговое окно открытия файла. Для этого сделайте двойной щелчок по соответствующему элементу OpenFileDialog на панели инструментов ToolBox. По существу, у диалогового окна Open File нет своего интерфейса, поэтому оно появляется не на форме, а под ней. Для того чтобы пользователь мог просматривать файлы, надо управлять диалоговым окном открытия файла с помощью его свойств и методов.

Теперь в форму надо добавить кнопку, с помощью которой можно выбрать файл. При выборе файла его имя будет отображено в текстовом поле. Настройте свойствакнопки как указано в таблице:

Свойство	*Значение*
Name	btnOpenFile
Location	8,8
Size	80,23
Text	Source:

Теперь сделайте двойной щелочек на кнопке и добавьте в событие Click такой текст:

```
OpenFileDialog1.InitialDirectory = "C:\"
OpenFileDialog1.Title = "Выбери файл"
```

Наконец, надо отобразить диалоговое окно Open File и когда в нем будет выбран файл, выполнить соответствующее действие. Метод ShowDialog диалогового окна open File работает так же, как и метод форм с тем же названием. Он возвращает результат, в котором указывается, что было выбрано пользователем в диалоговом окне.

```
If OpenFileDialog1.ShowDialog() <> Windows.Forms.DialogResult.Cancel Then
    txtSource.Text = OpenFileDialog1.FileName
Else
    txtSource.Text = ""
End If
```

Эта программа помещает указанные названия файлов в текстовое поле txtSource.

Добавьте кнопку **Cancel**.

Запустите проект нажатием F5 и щелкните на кнопке. Найдите созданную вами базу данных, выберите ее.

Диалоговое окно Save File

4. Создайте новое текстовое поле на форме с такими свойствами:

Свойство	*Значение*
Name	txtDestination
Location	95,40
Size	184,20
Text	(оставить пустым)

Теперь надо создать кнопку, при нажатии которой пользователь сможет выбрать файл для сохранения. Добавьте в форму новую кнопку и установите для нее такие свойства:

Свойство	*Значение*
Name	btnSaveFile
Location	8,40
Size	80,23
Text	Destination:

Добавьте диалоговое окно **Save File**.

Затем сделайте двойной щелчок по созданной кнопке btnSaveFile и добавьте в ее событие Click такой текст:

```
SaveFileDialog1.Title = "Specify Destination Filename"
SaveFileDialog1.OverwritePrompt = True
```

Наконец, надо написать последнюю часть программы, которая помещает указанное имя файла в текстовое поле txtDestination:

```
If SaveFileDialog1.ShowDialog() <> Windows.Forms.DialogResult.Cancel Then
    txtDestination.Text = SaveFileDialog1.FileName
End If
```

Запустите проект, проверьте, как работает диалоговое окно. Сохраните созданную вами базу данных под именем firma_copy.mdb.

5.Скопировать созданную базу данных в другую папку

Для начала добавьте функцию для проверки существования файла в форму класса:

```
Public Class Form1
    Private Function SourceFileExists() As Boolean
        If Not (System.IO.File.Exists(txtSource.Text)) Then
            MsgBox("Такой файл не существует", MsgBoxStyle.Exclamation)
        Else
            SourceFileExists = True
        End If
    End Function
```

6. Теперь добавим в форму новую кнопку. Если щелкнуть по этой кнопке, то файл, указанный в текстовом поле Source будет скопирован под именем, которое указывается в текстовом поле Destination. Установите свойства этой кнопки, как показано в таблице:

Свойство	*Значение*
Name	btnCopyFile
Location	96, 80
Size	75, 23
Text	Copy

Добавьте такой текст:

```
If Not (SourceFileExists()) Then Exit Sub
System.IO.File.Copy(txtSource.Text, txtDestination.Text)
MsgBox("The file has been successfully copied")
```

7. Запустите проект нажатием клавиши <F5>. Для того чтобы протестировать его, сделайте следующее:

1. Щелкните на кнопке Source и выделите файл, содержащий созданную вами базу данных.
2. Для того чтобы появилось диалоговое окно Save File, щелкните на кнопке Destination. Укажите в текстовом поле File Name имя firma_copy.mdb и нажмите кнопку Save. Если будет задан вопрос, заменять ли файл, выберите No и измените имя файла. Не следует использовать имя существующего файла.
3. Для того чтобы скопировать файл, щелкните на кнопке Copy.

После того как появится окно сообщения, в котором подтверждается копирование файла, его нужно найти и открыть с помощью СУБД. Сохраните свою работу.

Контрольные вопросы

1. Для чего предназначен управляющий элемент Open File?
2. Открывает ли файл управляющий элемент Open File?
3. Где появляется управляющий элемент Open File, есть ли у него свой интерфейс?
4. Опишите свойство OverwritePromt диалогового окна Save File
5. Назначение метода Copy (), его аргументы
6. Что необходимо сделать перед тем, как выполнять с файлом операцию копирования?
7. Для чего предназначен метод Exists (), каково его значение?
8. Когда создаются переменные типа Boolean какое значение присваивается им по умолчанию?

Лабораторная работа № 17

Тема «Реализация алгоритмов обработки числовых данных. Отладка приложения»

Цель работы: ознакомиться с инструментальными средствами моделирования деловых процессов, их классификацией и примерами.

Теоретическая часть

Процесс выявления причин обнаруженной ошибки, определение места (локализация) ошибки в программе и исправление ошибочно реализованного программного кода обычно называется *отладкой*.

Как правило, при отладке существует некоторая предварительная стадия, во время которой программист выдвигает те или иные предложения о причинах ошибочной работы и проводит визуальный анализ (*инспекцию*) программного кода.

Если при инспекции кода выявить ошибки не удается, далее наступает основной способ отладки – *отладочное выполнение программы* (или *трассировка*), в ходе которого работа программы может быть приостановлена для просмотра значений тех или иных переменных программы с целью обнаружения ситуаций, когда эти значения не соответствуют предполагаемых. Тем самым, задача трассировки – обнаружения информационных признаков проявления ошибки.

Рассмотрим далее возможности среды MS VS .NET для обеспечения трассировки программ при поиске и исправления ошибок.

Пошаговое выполнение программы

Для выполнения программы в пошаговом режиме (в режиме трассировки) используются четыре команды, которые доступны из меню **Debug**, панели инструментов **Debug** и клавиш быстрого вызова:

– Команда **Step Info** (клавиша **F11**) обеспечивает последовательное, строка за строкой, выполнение программного кода программы (включая содержимое вызываемых методов),

– Команда **Step Over** (клавиша **F10**) обеспечивает, как и предшествующая команда **Step Info**, последовательное выполнение программы, но при этом вызов методов рассматривается как один неделимый шаг (т.е. без перехода внутрь вызываемых методов),

– Команда **Step Out** (клавиша **Shift+F11**) обеспечивает выполнение всех оставшихся строк программного кода текущего выполняемого метода без останова, позволяя выполнить быстрый переход в последнюю точку вызова,

– Команда **Run to Cursor** (клавиша **Ctrl+F10**) обеспечивает выполнение без останова программного кода между текущей строки останова и позицией курсора (в зависимости от настроек параметров среды MS VS .NET данная команда может отсутствовать в пункте меню **Debug**).

Удобным средством указания точек останова процесса выполнения программы является использование *контрольных точек* (breakpoints). Для определения контрольной точки необходимо щелкнуть мышкой на вертикальной полосе слева от нужной строки программного кода; повторный щелчок отменяет установки контрольной точки. В ходе выполнения программы при попадании на контрольную точку происходит останов; для продолжения работы необходимо выполнить команду **Continue** пункта меню **Debug**. **Практическая часть:**

Наблюдение значений переменных

Для наблюдения значений переменных в момент останова выполнения программы достаточно расположить указатель мыши на имени переменной – в результате значение переменной появится в виде всплывающей подсказки.

Дополнительная возможность для наблюдения значений переменных состоит в использовании специальных окон наблюдения:

– Окно **Autos** отображает значения всех переменных, используемых в текущей и предшествующей строках точки останова программы; в окне отображаются названия переменных, их тип и значения; окно **Autos** обычно располагается в нижней левой части экрана (см. рис. 1.8) и для его высветки необходимо щелкнуть мышью на ярлычке с названием окна;

– Окно **Locals** отличается от предшествующего окна **Autos** тем, что отображает значения всех переменных текущей области видимости (т.е. переменных текущего выполняемого метода или его локального блока);

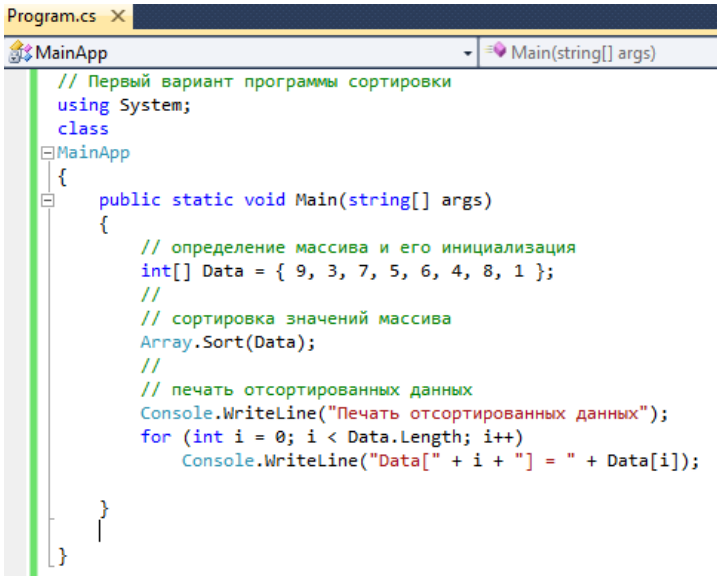
– Окна **Watch** (таких окон в момент выполнения 4) отличаются тем, что состав отображаемых в них переменных может формироваться непосредственно программистом. Для высветки нужного окна нужно последовательно выполнить команды **Debug\Windows\Watch\Watch <N>**, где N есть номер высвечиваемого окна. Для добавления переменной в окно для наблюдения нужно указать мышкой необходимую переменную, нажать правую кнопку мыши и появившемся контекстном меню выполнить команду **Add Watch** (такая же команда может иметься в пункте меню **Debug**, ее наличие в меню зависит от настроек параметров среды MS VS .NET). Удобный способ добавления переменных в окна наблюдения состоит в использовании техники "Взять и перенести" (выделить имя переменной, нажать левую кнопку мыши и, не отпуская ее, переместить указатель мыши в окно наблюдения, после чего отпустить конку мыши). Для удаления переменных из окна наблюдения достаточно выделить соответствующую строку и нажать клавишу <**Delete**>;

– Близким по назначению к окнам **Watch** является окно **Quick Watch**, которое дополнительно позволяет изменять значения наблюдаемых переменных; для высветки окна необходимо выделить нужную переменную и выполнить команду **Quick Watch** пункта меню **Debug**.

Кроме перечисленных окон, может быть использовано окно **this** для наблюдения за значениями полей объекта, метод которого выполняется в текущий момент времени, а также окно **Call Stack**, в котором отображается последовательность вызова методов, приведшая к обращению к текущему исполняемому методу.

Практическая часть

1. Создать консольное приложение (пример рассматривается для C#)для сортировки массивов методом "пузырька".



Для наблюдения итогов выполнения программы окно вывода надо задержать – это можно обеспечить, например, при помощи вызова процедуры вводаперед завершением метода **Main**.

```
// приостановка окна вывода
Console.ReadLine();
```

2. Выполним расширение первоначального простого варианта программы для автоматического заполнения сортируемого массива набором значений, генерируемых при помощи датчика случайных чисел, а также выполним реализацию одного из методов упорядочивания данных - алгоритма пузырьковой сортировки.

Добавление нового метода класса

Создадим метод **DataGenerator** для генерации сортируемого набора значений при помощи датчика случайных чисел:

```
// генератор данных
static void DataGenerator(int[] Vals)
{
    // Random - класс для генерации случайных чисел
    Random aRand = new Random();
    // заполнение массива
    for (int i = 0; i < Vals.Length; i++)
        Vals[i] = aRand.Next(100);
}
```

Дадимкраткиепояснениядляметода**DataGenerator**:

– Метод описан как **static** – как результат, метод может быть вызван по имени класса;

– Метод имеет входной параметр – массив **Vals**, который и должен быть заполнен генерируемым набором значений;

– Для генерации значений используется объект класса **Random** (следует обратить внимание, как происходит создание этого объекта);

– Для генерации следующего случайного значения используется метод **Next**; параметр метода задает для датчика случайных чисел максимально-возможное генерируемое значение (минимально-возможное значение равно нулю).

Набор метода **DataGenerator** целесообразно выполнить перед методом **Main**.

При наличии метода для генерации значении участок программного кода по формированию массива должен быть заменен на следующий фрагмент:

```
// определение массива и его инициализация
```

```
// определение массива и его инициализация
const int N = 10;
int[] Data = new int[N];
DataGenerator(Data);
```

3. Расширим создаваемую учебную программу собственно реализованным методом упорядочивания данных - выберем для этого хорошо известный и сравнительно простой алгоритм пузырьковой сортировки. Метод основывается на базовой операции "сравнить и переставить" (compare-exchange), состоящей в сравнении той или иной пары значений из сортируемого набора данных и перестановки этих значений, если их порядок не соответствует условиям сортировки

```
// операция "сравнить и переставить"
if ( a[i] > a[j] ) {
temp = a[i];
a[i] = a[j];
a[j] = temp;
}
```

Последовательно применив данную процедуру сравнения всех соседних элементов в результате прохода по не упорядочиваемому набору данных в последнем (верхнем) элементе массива оказывается максимальное значение ("всплывание пузырька"); далее для продолжения сортировки этот уже упорядоченный элемент может быть отброшен и действия алгоритма следует повторить

```
// пузырьковая сортировка
for ( i=1; i<n; i++ )
for ( j=0; j<n-i; j++ )
<сравнить и переставить элементы (a[j],a[j+1])>
}
```

В результате реализация алгоритма пузырьковой сортировки может быть выполнена следующим образом:

```
// метод пузырьковой сортировки
static void BubbleSort(int[] Vals)
{
    int temp;
    for (int i = 1; i < Vals.Length; i++)
        for (int j = 0; j < Vals.Length - i; j++)
            // сравнить и переставить элементы

             if (Vals[j] > Vals[j + 1])
            {
                temp = Vals[j];
                Vals[j] = Vals[j + 1];
                Vals[j + 1] = temp;
            }
}
```

В методе **Main** сохраним сортировку стандартным методом (для последующего сравнения результатов работы). Как результат, для исходного массива необходимо создать копию

```
// создание копии массива
int[] Data2 = new int[N];
Data.CopyTo(Data2, 0);
```

Как видно из примера, создание копии обеспечивается методом **CopyTo** (значение 0 указывает, что копирование необходимо выполнить с нулевого элемента массива). Разместить данный код следует сразу после заполнения исходного массива (после вызова метода **DataGenerator**).

Для использования пузырьковой сортировки можно продублировать уже имеющийся в Main программный код (заменив массив Data на массив Data2)

```
    // сортировка при помощи пузырьковой сортировки
    BubbleSort(Data2);
    //
    // печать отсортированных данных
    Console.WriteLine("Печать данных после пузырьковой сортировки");
for (int i=0; i<Data2.Length; i++)
    Console.WriteLine("Data2["+i+"] = " + Data2[i]);
```

Полный вариант программы подготовлен. Далее необходимо построить сборку и выполнить несколько экспериментов для проверки правильности работы алгоритма сортировки.

Проверить визуально правильность работы можно только для небольших массивов данных и для ограниченного набора контрольных примеров. Крайне желательно пытаться автоматизировать процесс проверки результатов – так, для проверки совпадения результатов сортировки можно подготовить специальный программный код:

```
    // автоматическая проверка результатов сортировки
    bool IsEqual = true;
    for (int i=0; i<Data2.Length; i++)
    if ( Data[i] != Data2[i] ) IsEqual = false;
    if ( IsEqual ) Console.WriteLine("Результатысовпадают");
    else Console.WriteLine("Ошибки в пузырьковой сортировке");
```

(при организации такой проверки делается предположение, что метод стандартной сортировки работает правильно).

Выполним еще один дополнительный шаг – определим время, которое затрачивается выбранными методами сортировки на выполнение. Окружим для этого вызовы методов сортировки операторами получения системного времени

```
    // сортировка значений массива
    long time;
    time = Environment.TickCount;
    Array.Sort(Data);
    time = Environment.TickCount - time;
    Console.WriteLine("Время стандартной сортировки: "
    + time.ToString() +" msecs");
    //
    // сортировка при помощи пузырьковой сортировки
    time = Environment.TickCount;
    BubbleSort(Data2);
    time = Environment.TickCount - time;
    Console.WriteLine("Времяпузырьковойсортировки: "
+ time.ToString() +" msecs");
```

Пример выполнения отладки

Перечисленных набор средств среды разработки MS VS .NET достаточно для организации быстрой и эффективной отладки. Для успешного освоения этих средств необходима понимание принципов отладки и практика по их использованию для поиска и исправления ошибок при разработке достаточно сложных программ.

Приведем пример проведения процесса отладки с использованием созданной программы. Внесем ошибку в программу – заменим оператор

```
Vals[j+1] = temp;
```

в методе пузырьковой сортировки **BubbleSort** на оператор

```
Vals[j] = temp;
```

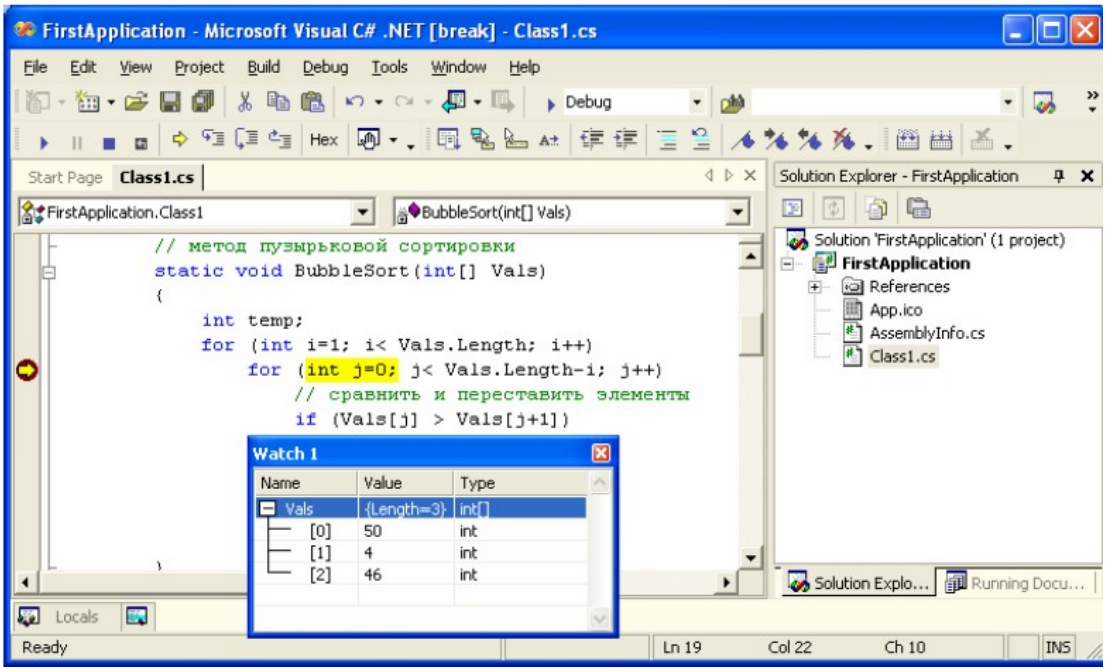
(такой прием, конечно, дает достаточно слабое представление о процедуре отладки – больший эффект можно получить, если внесение ошибки произвольного вида (!) будет выполнено кем то другим).

Выполнение программы с внесенной ошибкой приведет к тому, что результирующий массив не будет являться отсортированным – возможная схема отладки может состоять в следующем:

– Можно попытаться проверить, наблюдается ли ошибочный эффект при меньшем размере сортируемого набора данных; установим для этого значения константы N=3 и повторим выполнение программы – массив по прежнему остается неотсортированным;

– Попытаемся определить причину ошибки – проблема может состоять или в неправильной работе алгоритма сортировки или ошибочными являются операторы вывода значений массива; однако вывод результатов стандартной сортировки сработал правильно и, кроме того, автоматическая проверка результатов сортировки тоже подтвердила, что результат пузырьковой сортировки является неправильным; как результат, можно сделать вывод, что ошибки содержатся в методе пузырьковой сортировки **BubbleSort**;

– Установим контрольную точку на внутреннем операторе цикла в методе **BubbleSort** (см. рис.) и запустим программу на выполнение; после останова добавим массив **Vals** в окно наблюдения **Watch 1** и запомним исходное значение сортируемого массива;



–Выполнение внешней итерации алгоритма пузырьковой сортировки должно привести к "всплыванию" максимального значения в последний элемент массива; выполним команду **Continue** пункта меню **Debug** – результат выполнения не привел к желаемому эффекту, состояние сортируемого массива не изменилось и можно сделать вывод, что внешняя итерация алгоритма сортировки работает неправильно;

–Поскольку сортируемый массив состоит только из трех элементов, следующая итерация алгоритма сортировки должна оказаться последней; в ходе ее выполнения значения двух первых элементов должно поменяться местами (на примере значений из рис.); выполним трассировку – нажмем дважды клавишу **F10** и перейдем на операторы перестановки значений, т.е. сравнение пары значений выполняется корректно; однако последующее выполнение операторов перестановки не приводит к нужному результату – отсюда следует, что ошибка содержится в алгоритме перестановки пары значений; анализ данного участка программного кода позволяет определить, что индекс элемента массива в последнем операторе должен быть **j+1**; для исправления ошибки вносим необходимые изменения и программа начинает работать правильно.

Контрольные вопросы

1. Что называется отладкой программы?
2. Что такое инспекция программного кода?
3. Назовите основной способ отладки?

Лабораторная работа № 18

Тема «Реализация алгоритмов поиска. Отладка приложения»

Цель работы: ознакомиться с реализацией алгоритма поиска при создании информационной системы.

Теоретическая часть

Когда компания Jones, Inc. приняла решение создать новую систему обработки данных, она уже имела прежнюю систему на основе базы данных

Access, в которой директор компании Брэд Джонс хранил данные о заказах. Для директора очевидны преимущества перехода к новой системе на основе СУБД SQL Server, но прежде всего он хотел бы гарантировать сохранность унаследованных данных. Поэтому было решено вносить все изменения постепенно, особенно для сохранения данных компании за прошлые годы.

Итак, Джонс хотел бы создать новую систему на основе SQL Server, но иногда использовать старые данные из базы данных Access до тех пор, пока не будет закончена работа над новой системой. Поэтому часть данных должна храниться в одной базе данных, а часть — в другой. Две базы данных должны совместно использовать и даже объединять свои данные, хотя они относятся к разным типам баз данных.

К счастью, это требование легко удовлетворяется в модели ADO.NET. Для объекта DataSet совершенно не имеет значения, откуда, то есть из каких источников, берутся данные.

Несмотря на то, что DataSet является фрагментом (кэшем) базы данных, он не имеет постоянной фактической связи с первоисточником. Объект DataSet — это контейнер, заполняемый информацией другим объектом — адаптером, данных - DataAdapter, который взаимодействует с первоисточником через SQL-запросы или хранимые процедуры. Один объект DataSet может взаимодействовать с несколькими объектами DataAdapter, каждый из которых обеспечивает наполнение данными таблиц контейнера.

Поскольку объект DataSet непосредственно не связан с источником данных, существует хорошая предпосылка для интеграции (объединения) данных, которые поступают из разных источников. Например, часть информации в DataSet может поступить из базы данных центрального офиса компании, часть из базы данных удаленного филиала, или вообще не из базы данных, а из другого источника, например, из электронной таблицы. Как только данные поступят в контейнер DataSet, пользователь может работать с ними как с единым информационным массивом, используя свойства и методы одного объекта, и абсолютно независимо от оригинального источника данных.

Создадим проект, выводящий информацию о клиенте по коду туристической путевки: в текстовое поле пользователь будет вводить код путевки (информация из базы Access таблицы Путевки), в поле со списком будет выводиться связанная информация о клиенте, который приобрел путевку (информация из базы SQL таблицы Информация о туристах), в сетке информация из таблицы Путевки.

Для работы используйте базы данных BDTur_firm.mdb и BDTur_firmSQL.mdf

Практическая часть

1. Создайте на рабочем столе папку с именем, созвучным вашей фамилии

Создание нового проекта и подключения данных

2. После запуска VisualStudio.Net создайте новый проект, выбрав шаблон WindowsApplication. Назовите проект созвучно вашей фамилии.

В окне ***Свойств*** формы укажите значение *frmTurFirm* для свойства (Name) и значение *ShowTable* для свойства Text формы Form1.

3. В верхнем левом углу формы создайте текстовое поле и укажите значение *txtCode* для его свойства (Name)

4. Создайте кнопку и укажите значение *btnFinds* для ее свойства (Name) и значение *Find* для ее свойства Text

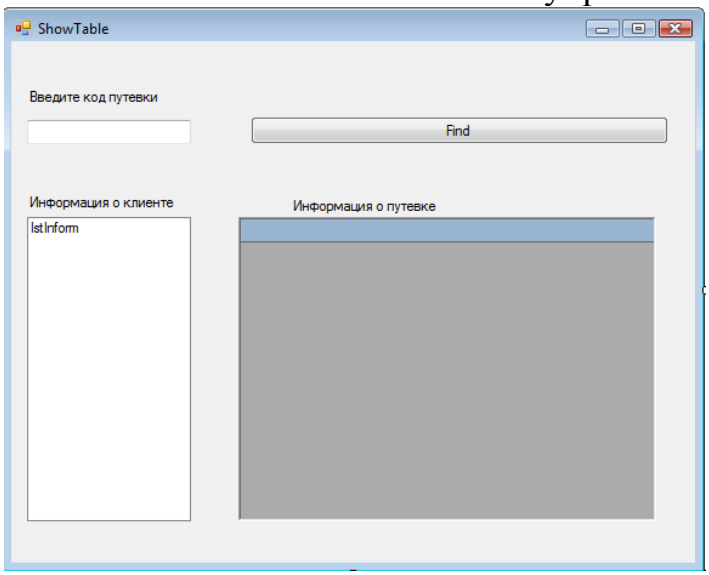
5. Создайте поле со списком и укажите значение *lstInform* для его свойства (Name).

6. В правой части формы создайте сетку данных (элемент DataGrid) и укажите значение grdTable для ее свойства (Name).

7. Над текстовым полем разместите метку с текстом «Введите код путевки», а над списком метку с текстом «Информация о клиенте»

8. Над сеткой разместите метку с текстом «Информация о путевке»

9. Расположите все эти элементы управления так, как показано на рисунке



В окне ***Обозреватель серверов подключитесь к базам данных.***

Обратите внимание, что это две базы с разными расширениями: одна база данных формата .mdb, другая – mdf.

10. Перетащите в форму из панели элементов управления компонент DataSet (выбрав Нетипизированный набор данных) и укажите значение *dsTables* для его свойства (Name).

11. Создайте новый объект SqlDataAdapter для таблицы **Информация о туристах** базы данных (*она лежит в папке Базы_для_15*) **BDTur_firmSQL.mdf**СУБД SQL Server, перетаскивая его из панели элементов управления. После запуска программы-мастера настройки адаптера данных выберите подключение к базе данных BDTur_firmSQL.mdf. Используйте инструкции SQL.

12. В диалоговом окне ***Создание инструкцийSQL*** укажите команду SELECT * FROM Информация о туристах для загрузки данных в объект – набор данных.

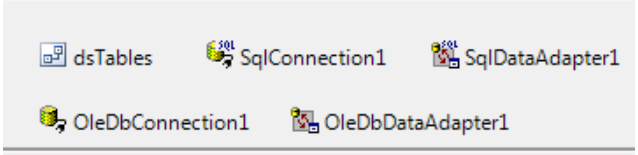
13. Для таблицы Путевки, которая находится в базе данных BDTur_firm.mdb Access создайте объект OledbDataAdapter, перетаскивая его из панели элементов управления.

14. После запуска программы-мастера выберите подключение к файлу базы данных BDTur_firm.mdb

15. Затем в диалоговом окне ***Выбор типа команд*** выберите переключатель Использовать инструкции SQL

16. В диалоговом окне Создание инструкций SQL укажите команду SELECT * FROM Путевки для загрузки данных в объект – набор данных.

После этих действий под формой должны находиться следующие элементы:



17. Откройте код формы и введите:

```
Imports System
Imports System.Data
Imports System.Data.SqlClient
Imports System.Data.OleDb
Public Class frmTurFirm

End Class
```

18. Затем в определении класса формы *frmTurFirm* введите код

```
Public Class frmTurFirm
    Private dvOrders As New DataView()
```

19. В событии Load формы введите следующий код

```
Public Class frmTurFirm
    Private dvOrders As New DataView()
    Private Sub frmTurFirm_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        Dim rel As DataRelation
        ' Вставка таблиц в набор данных dsTables.
        SqlDataAdapter1.Fill(dsTables, "Информация о туристах")
        OleDbDataAdapter1.Fill(dsTables, "Путевки")
        ' Создание отношения между таблицами.
        rel = dsTables.Relations.Add("relInfTur", dsTables.Tables("Информация о туристах")_
.Columns("Код туриста"), dsTables.Tables("Путевки").Columns("Код путевки"))
        ' Указание первичного ключа для таблицы Информация о туристах.
        Dim pk(0) As DataColumn
        pk(0) = dsTables.Tables("Информация о туристах").Columns("Код туриста")
        dsTables.Tables("Информация о туристах").PrimaryKey = pk
        'Указание сортировки по умолчанию для метода Find.
        dsTables.Tables("Информация о туристах").DefaultView.Sort = "Код туриста"
    End Sub
```

20. В событии Click кнопки Find введите следующий код:

```
    Private Sub btnFind_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnFinds.Click
        Dim RowNum As Integer
        Dim dvRow As DataRowView
        Dim i As Integer
        If IsNumeric(txtCode.Text) Then
            RowNum = dsTables.Tables("Информация о туристах").DefaultView.Find(txtCode.Text)
            If RowNum <> -1 Then
                dvRow = dsTables.Tables("Информация о туристах").DefaultView(RowNum)
                ' Вставка в поле со списком имен полей из таблицы Информация о туристах.
                lstInform.Items.Clear()
                For i = 0 To dsTables.Tables("Информация о туристах").Columns.Count - 1
                    lstInform.Items.Add(dvRow.Item(i))
                Next
                grdTable.CaptionText = "Путевки" & txtCode.Text
                ' Извлечение связанных дочерних записей для
                ' указанного клиента таблицы Информация о туристах.
                dvOrders = dvRow.CreateChildView("relInfTur")
                grdTable.DataSource = dvOrders
            Else
                MessageBox.Show("Клиент  не найден - Попробуйте еще раз.")
                txtCode.Clear()
            End If
        Else
            Beep()
        End If
    End Sub
End Class
```

Все параметры источников данных задаются в подпрограмме frmTurFirm_Load. Затем создаются две таблицы для набора данных DataSet и отношение DataRelation между ними. Наконец, задаются значения свойств PrimaryKey и Sort для таблицы Информация о туристах и его представления по

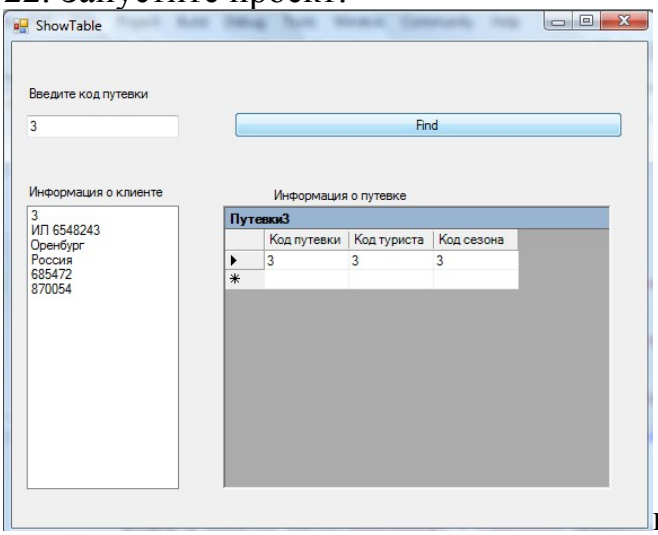
умолчанию Defaultview, чтобы для поиска записей можно было использовать метод Find.

Щелчок на кнопке Find приводит к выполнению нескольких операций, которые определены в коде подпрограммы btnFind_Click. После проверки числового типа значения в поле txtCode начинается поиск этого значения в представлении DefaultView таблицы Информация о туристах. Если указанное значение найдено, то в поле со списком DataRowView будут показаны значения всех полей искомой записи с заданным идентификатором из таблицы Информация о туристах, а в сетке данных справа будут показаны ее дочерние записи из таблицы Путевки.

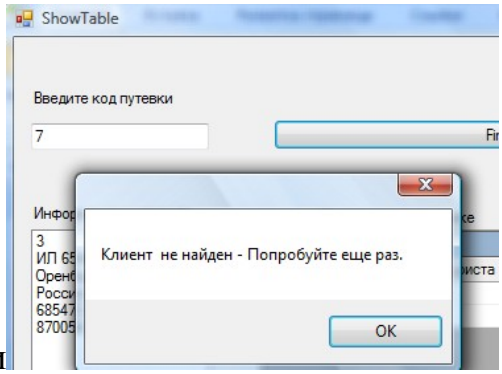
Обратите внимание, что в данном примере создано отношение между двумя таблицами из баз данных совершенно разного типа!

21. Добавьте кнопку выхода

22. Запустите проект:



или



23. Создайте инсталляционный пакет.

При этом в папку ApplicationFolder добавьте файлы используемых в проекте баз данных (в контекстном меню выберите команду Add - File). Это позволит при инсталляции установить и базы данных.

Контрольные вопросы

1. Опишите, как создается набор данных в проекте, как производится его заполнение?

2. Какими таблицами заполняется набор данных проекта, из каких баз данных? Укажите программный код для заполнения набора данных

3. Какие операции определены в коде подпрограммы btnFind_Click? Опишите их действия

4. Привяжите сетку данных к набору данных (как это сделать?), что изменится?

5. Объясните работу фрагмента кода:

```
Else
    MessageBox.Show("Клиент  не найден - Попробуйте еще раз.")
    txtCode.Clear()
End If
```

Лабораторная работа № 19

Тема «Реализация обработки табличных данных. Отладка приложения**»**

Цель работы: ознакомиться с инструментальными средствами моделирования деловых процессов, их классификацией и примерами.

Теоретическая часть

При достаточно сложной обработке информации невозможнообойтись без структурных типов данных.

Массив – это таблица элементов, имеющих одно имя, ноотличающихся значением индекса – номера элемента. Одномерныймассив объявляется с помощью записи следующей формы:

```
Dim имя_массива(макс_индекс) As тип
```

Здесь ***тип*** – это базовый тип данных каждого элемента массива.

С элементами массива можно выполнять все те же действия, чтои с простыми переменными указанного типа. Существенное отличиезаключается в том, что номер элемента может указываться спомощью числовой переменной, а также с помощью арифметическоговыражения. Это позволяет строить сложные алгоритмы обработкиданных.

```
Dim a(10) As Integer
Dim n As Integer
A(0) = 10
A(1) = 7
n = 2
a(n) = a(0)+a(n-1)
```

Значения элементов массива могут быть заданы с помощьюоператора присваивания, операцией ввода данных с клавиатуры илииз файла, а также инициализацией в момент создания, например,массива вещественных чисел:

```
Dim b() As Double = {0.0,1.0,2.0,3.0,4.0}
```

Количество элементов здесь не указывается явно, а определяетсяпо количеству чисел, заданных при инициализации. Допускаетсядругая, несколько избыточная форма объявления массива:

```
Dim b2() As Double = New Double() {0.1, 1.3, 2.5, 3.7, 4.9}
```

В еще одном варианте объявления массив создается в однойстроке, а инициализируется в другой:

```
Dim b3() As Double
b3 = New Double() {0.01, 1.02, 2.03, 3.04, 4.05}
```

Практическаячасть

В примере создается один целочисленный и 3 вещественныхмассива из 5 элементов. В целочисленный массив заносятсяслучайные числа от 10 до 99. Вещественные массивы инициируютсянепосредственно при объявлении. Затем значения элементов массивавыводятся на экран.

Пример 3.1

```
Module Module1
Sub Main()
Dim A(4) As Integer
Dim b() As Double = {0.02, 1.04, 2.06, 3.08, 4.1}
Dim b2() As Double = New Double() {0.1, 1.3, 2.5, 3.7, 4.9}
```

```
Dim b3() As Double
b3 = New Double() {0.01, 1.02, 2.03, 3.04, 4.05}
For I As Integer = 0 To 4
A(I) = 10 + Int(90*Rnd)
Next I
For i As Integer = 0 To 4
Console.Write(A(i))
Console.Write(" " & b(i).ToString("00.00"))
Console.Write(" ")
Console.Write("{0:R2}", b2(i))
Console.Write(" ")
Console.Write("{0:F2}", b3(i))
Console.WriteLine()
Next
Console.ReadLine()
End Sub
EndModule
```

При выводе чисел использовано 3 способа форматирования.

Первый – ***b(i).ToString("00.00")*** означает, что число выводится собязательными 2 знаками до запятой и 2 знаками после запятой.

Второй и третий форматы похожи – они означают, что числовыводится с обязательными 2 знаками после запятой. Числаотделяются друг от друга пробелами.

Задача

Сформировать два массива и записать их содержимое втекстовый файл. В таблице указаны типы данных, которые должныбыть использованы в программе.

Вариант	Массив 1	Массив 2
1	char	integer
2	char	long
3	char	single
4	char	double
5	char	short
6	byte	integer
7	byte	long
8	byte	single

Вариант	Массив 1	Массив 2
9	byte	double
10	single	integer
11	single	long
12	single	short
13	double	integer
14	double	long
15	double	short

Порядок выполнения лабораторной работы

Создать новый проект – консольное приложение.

Добавить необходимый программный код.

Компилировать программу.

Тестировать программу.

Создать документ с описанием работы программы.

Контрольные вопросы

1. Что называется табличными данными?

2. Какие операторы VB.Netможно использовать для работы с табличными данными?
3. Назовите основной способ отладки?

Лабораторная работа № 20

Тема «Разработка и отладка генератора случайных символов»

Цель работы: ознакомиться с инструментальными средствами моделирования деловых процессов, их классификацией и примерами.

Теоретическая часть

Рассмотрим создание генератора символов на основе Марковских процессов.

Марковский процесс — случайный процесс, эволюция которого после любого заданного значения временного параметра *t* не зависит от эволюции, предшествовавшей *t*, при условии, что значение процесса в этот момент фиксировано.

Марковская цепь — частный случай марковского процесса, когда пространство его состояний дискретно (т.е. не более чем счетно).

Первый пример предельно прост. Используя предложение из детской книжки, мы освоим базовую концепцию цепи Маркова, а также определим, что такое в нашем контексте **корпус, звенья, распределение вероятностей и гистограммы**. Несмотря на то, что предложение приведено на английском языке, суть теории будет легко уловить.

One fish two fish red fish blue fish

Это предложение и есть **корпус**, то есть база, на основе которой в дальнейшем будет генерироваться текст. Оно состоит из восьми слов, но при этом уникальных слов только пять — это **звенья** (мы ведь говорим о марковской *цепи*). Для наглядности окрасим каждое звено в свой цвет:

One fish two fish red fish blue fish

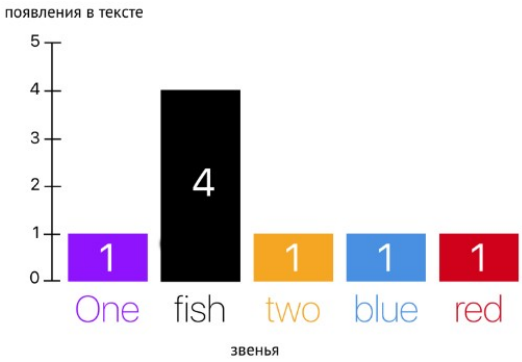
И выпишем количество появлений каждого из звеньев в тексте:

One : 1
fish : 4
two : 1
red : 1
blue : 1

На картинке выше видно, что слово *«fish»* появляется в тексте в 4 раза чаще, чем каждое из других слов (*«One», «two», «red», «blue»*). То есть вероятность встретить в нашем корпусе слово *«fish»* в 4 раза выше, чем вероятность встретить каждое другое слово из приведенных на рисунке. Говоря на языке математики, мы можем определить закон распределения случайной величины и вычислить, с какой вероятностью одно из слов появится в тексте после текущего. Вероятность считается так: нужно разделить число появлений нужного нам слова в корпусе на общее число всех слов в нем. Для слова *«fish»* эта вероятность — 50%, так как оно

появляется 4 раза в предложении из 8 слов. Для каждого из остальных звеньев эта вероятность равна 12,5% (1/8).

Графически представить распределение случайных величин можно с помощью **гистограммы**. В данном случае, наглядно видна частота появления каждого из звеньев в предложении:



Итак, наш текст состоит из слов и уникальных звеньев, а распределение вероятностей появления каждого из звеньев в предложении мы отобразили на гистограмме. Если вам кажется, что возиться со статистикой не стоит, прочитайте наш перевод, который вас переубедит. И, возможно, сохранит вам жизнь.

Суть определения

Теперь добавим к нашему тексту элементы, которые всегда подразумеваются, но не озвучиваются в повседневной речи — начало и конец предложения:

START One fish two fish red fish blue fish *END*

Любое предложение содержит эти невидимые «начало» и «конец», добавим их в качестве звеньев к нашему распределению:

START	:	1
One	:	1
fish	:	4
two	:	1
red	:	1
blue	:	1
END	:	1

Вернемся к определению, данному в начале статьи:

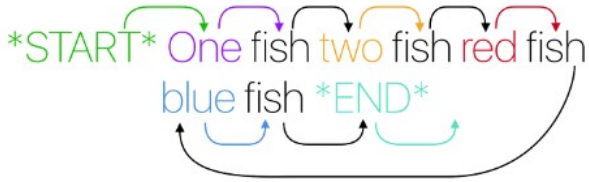
Марковский процесс — случайный процесс, эволюция которого после любого заданного значения временного параметра t не зависит от эволюции, предшествовавшей t, при условии, что значение процесса в этот момент фиксировано.

Марковская цепь — частный случай марковского процесса, когда пространство его состояний дискретно (т.е. не более чем счетно).

Мы моделируем процесс, в котором **состояние системы в следующий момент времени зависит только от её состояния в текущий момент, и никак не зависит от всех предыдущих состояний**.

Представьте, что перед вами **окно**, которое отображает только текущее состояние системы (в нашем случае, это одно слово), и вам нужно определить,

каким будет следующее слово, основываясь только на данных, представленных в этом окне. В нашем корпусе слова следуют одно за другим по такой схеме:



Таким образом, формируются пары слов (даже у конца предложения есть своя пара — пустое значение):

(*Start*, One)
(One, fish)
(fish, two)
(two, fish)
(fish, red)
(red, fish)
(fish, blue)
(blue, fish)
(fish, *END*)
(*END*, none)

Сгруппируем эти пары по первому слову. Мы увидим, что у каждого слова есть свой набор звеньев, которые в контексте нашего предложения **могут** за ним следовать:

(*Start*, One)
(One, fish)
(fish, two) (fish, red) (fish, blue) (fish, *END*)
(two, fish)
(red, fish)
(blue, fish)
(*END*, none)

Представим эту информацию другим способом — каждому звену поставим в соответствие массив из всех слов, которые могут появиться в тексте после этого звена:

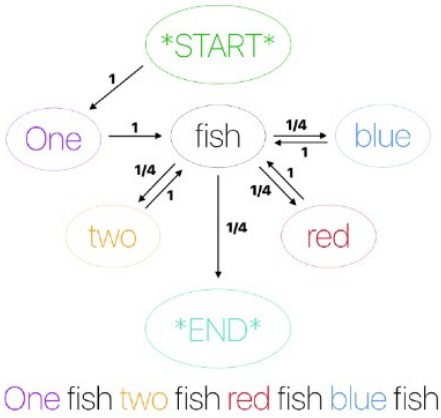
Start : [One]
One : [fish]
fish : [two, red, blue, *END*]
two : [fish]
red : [fish]
blue : [fish]
END : [none]

Разберем подробнее. Мы видим, что у каждого звена есть слова, которые **могут** стоять после него в предложении. Если бы мы показали схему выше кому-то еще, этот человек с некоторой вероятностью мог бы реконструировать наше начальное предложение, то есть корпус.

Пример. Начнем со слова *«Start»*. Далее выбираем слово *«One»*, так как по нашей схеме это единственное слово, которое может следовать за началом предложения. За словом *«One»* тоже может следовать только одно слово — *«fish»*. Теперь новое предложение в промежуточном варианте выглядит как *«One fish»*. Дальше ситуация усложняется — за *«fish»* могут с равной вероятностью в 25% идти слова *«two», «red», «blue»* и конец предложения *«End»*. Если мы

предположим, что следующее слово — *«two»*, реконструкция продолжится. Но мы можем выбрать и звено *«End»*. В таком случае на основе нашей схемы будет случайно сгенерировано предложение, сильно отличающееся от корпуса — *«One fish»*.

Мы только что смоделировали марковский процесс — определили каждое следующее слово только на основании знаний о текущем. Давайте для полного усвоения материала построим диаграммы, отображающие зависимости между элементами внутри нашего корпуса. Овалы представляют собой звенья. Стрелки ведут к потенциальным звеньям, которые могут идти за словом в овале. Около каждой стрелки — вероятность, с которой следующее звено появится после текущего:



Расширяем словарную базу

Возьмем еще четыре цитаты того же автора (также на английском, нам это не помешает):

«Today you are you. That is truer than true. There is no one alive who is you-er than you.»

«You have brains in your head. You have feet in your shoes. You can steer yourself any direction you choose. You're on your own.»

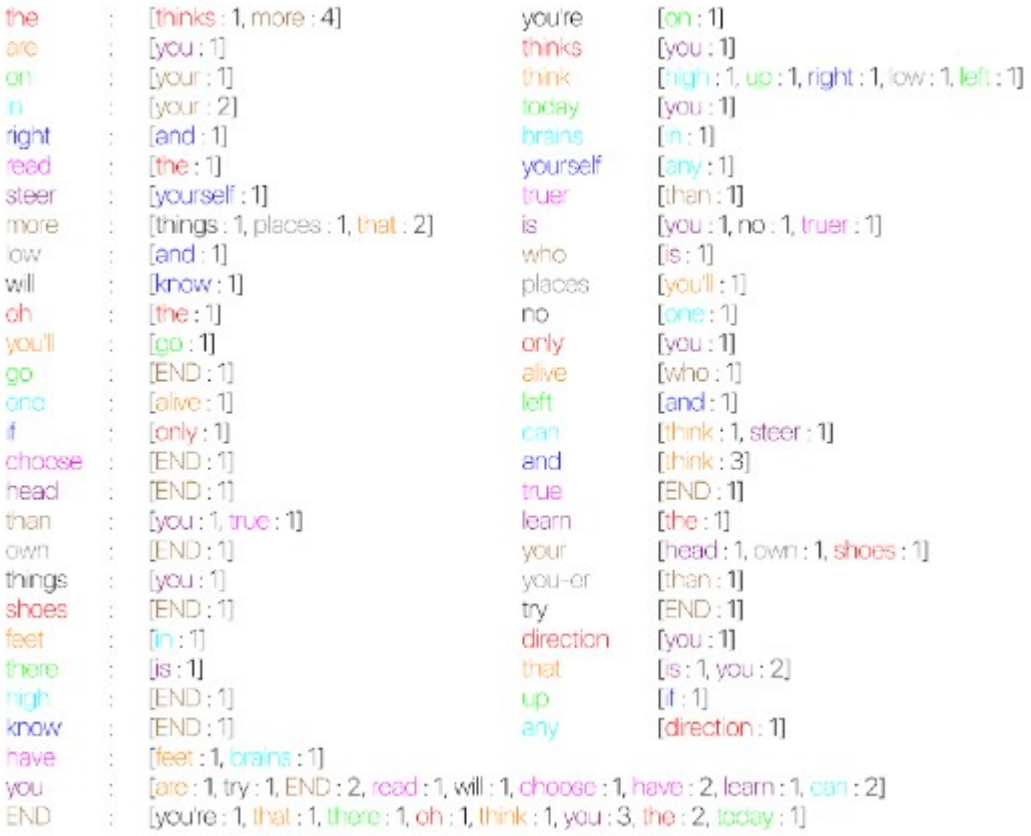
«The more that you read, the more things you will know. The more that you learn, the more places you'll go.»

«Think left and think right and think low and think high. Oh, the thinks you can think up if only you try.»

Сложность корпуса увеличилась, но в нашем случае это только плюс — теперь генератор текста сможет выдавать более осмысленные предложения. Дело в том, что в любом языке есть слова, которые встречаются в речи чаще, чем другие (например, предлог «в» мы используем гораздо чаще, чем слово «криогенный»). Чем больше слов в нашем корпусе (а значит, и зависимостей между ними), тем больше у генератора информации о том, какое слово вероятнее всего должно появиться в тексте после текущего.

Проще всего это объясняется с точки зрения программы. Мы знаем, что для каждого звена существует набор слов, которые могут за ним следовать. А также, каждое слово характеризуется числом его появлений в тексте. Нам нужно каким-то образом зафиксировать всю эту информацию в одном месте; для этой цели лучше всего подойдет словарь, хранящий пары «(ключ, значение)». В ключе словаря будет записано текущее состояние системы, то есть одно из звеньев корпуса (например, *«the»* на картинке ниже); а в значении словаря будет

храниться еще один словарь. Во вложенном словаре ключами будут слова, которые могут идти в тексте после текущего звена корпуса (*«thinks»* и *«more»* могут идти в тексте после *«the»*), а значениями — число появлений этих слов в тексте после нашего звена (слово *«thinks»* появляется в тексте после слова *«the»* 1 раз, слово *«more»* после слова *«the»* — 4 раза):



вложенный словарь в данном случае — это та же гистограмма, он помогает нам отслеживать звенья и частоту их появления в тексте относительно других слов. Надо заметить, что даже такая словарная база очень мала для надлежащей генерации текстов на естественном языке — она должна содержать более 20 000 слов, а лучше более 100 000. А еще лучше — более 500 000. Но давайте рассмотрим ту словарную базу, которая получилась у нас.

Цепь Маркова в данном случае строится аналогично первому примеру — каждое следующее слово выбирается только на основании знаний о текущем слове, все остальные слова не учитываются. Но благодаря хранению в словаре данных о том, какие слова появляются чаще других, мы можем при выборе принять **взвешенное решение**. Давайте разберем конкретный пример:

```
more : [things : 1, places : 1, that : 2]
```

То есть если текущим словом является слово *«more»*, после него могут с равной вероятностью в 25% идти слова *«things»* и *«places»*, и с вероятностью 50% — слово *«that»*. Но вероятности могут быть и все равны между собой:

```
think : [high : 1, up : 1, right : 1, low : 1, left : 1]
```

Работа с окнами

До настоящего момента мы с вами рассматривали только окна размером в одно слово. Можно увеличить размер окна, чтобы генератор текста выдавал более «выверенные» предложения. Это значит, что чем больше окно, тем меньше будет отклонений от корпуса при генерации. Увеличение размера окна соответствует

переходу цепи Маркова к более высокому порядку. Ранее мы строили цепь первого порядка, для окна из двух слов получится цепь второго порядка, из трех — третьего, и так далее.

Окно — это те данные в текущем состоянии системы, которые используются для принятия решений. Если мы совместим большое окно и маленький набор данных, то, скорее всего, каждый раз будем получать одно и то же предложение. Давайте возьмем словарную базу из нашего первого примера и расширим окно до размера 2:

(*Start*, one) : [fish : 1]
(one, fish) : [two : 1]
(fish, two) : [fish : 1]
(two, fish) : [red : 1]
(fish, red) : [fish : 1]
(red, fish) : [blue : 1]
(fish, blue) : [fish : 1]
(blue, fish) : [*END* : 1]

Расширение привело к тому, что у каждого окна теперь только один вариант следующего состояния системы — что бы мы ни делали, мы всегда будем получать одно и то же предложение, идентичное нашему корпусу. Поэтому, чтобы экспериментировать с окнами, и чтобы генератор текста возвращал уникальный контент, запаситесь словарной базой от 500 000 слов.

Практическая часть

Реализация на Python

Структура данных Dictogram

Dictogram (dict — встроенный тип данных словарь в Python) будет отображать зависимость между звеньями и их частотой появления в тексте, то есть их распределение. Но при этом она будет обладать нужным нам свойством словаря — время выполнения программы не будет зависеть от объема входных данных, а это значит, мы создаем эффективный алгоритм.

```
import random

classDictogram(dict):
def__init__(self, iterable=None):
# Инициализируем наше распределение как новый объект класса,
        # добавляем имеющиеся элементы
super(Dictogram, self).__init__()
        self.types =0  # число уникальных ключей в распределении
        self.tokens =0  # общее количество всех слов в распределении
if iterable:
self.update(iterable)

def update(self, iterable):
        # Обновляем распределение элементами из имеющегося
# итерируемогонабораданных
for item in iterable:
if item in self:
self[item]+=1
                self.tokens +=1
else:
```

```
self[item]=1
                self.types +=1
                self.tokens +=1

defcount(self, item):
# Возвращаем значение счетчика элемента, или 0
if item in self:
return self[item]
return0

defreturn_random_word(self):
        random_key =random.sample(self,1)
        # Другойспособ:
        # random.choice(histogram.keys())
return random_key[0]

defreturn_weighted_random_word(self):
# Сгенерировать псевдослучайное число между 0 и (n-1),
# где n –общеечислослов
        random_int =random.randint(0, self.tokens-1)
index=0
        list_of_keys =self.keys()
# вывести 'случайный индекс:', random_int
for i inrange(0, self.types):
index+= self[list_of_keys[i]]
            # вывестииндекс
if(index > random_int):
                # вывести list_of_keys[i]
return list_of_keys[i]
```

В конструктор структуре Dictogram можно передать любой объект, по которому можно итерироваться. Элементами этого объекта будут слова для инициализации Dictogram, например, все слова из какой-нибудь книги. В данном случае мы ведем подсчет элементов, чтобы для обращения к какому-либо из них не нужно было пробегать каждый раз по всему набору данных.

Мы также сделали две функции для возврата случайного слова. Одна функция выбирает случайный ключ в словаре, а другая, принимая во внимание число появлений каждого слова в тексте, возвращает нужное нам слово.

Структура цепи Маркова

```
from histograms import Dictogram

defmake_markov_model(data):
    markov_model =dict()

for i inrange(0,len(data)-1):
if data[i]in markov_model:
# Просто присоединяем к уже существующему распределению
markov_model[data[i]].update([data[i+1]])
else:
            markov_model[data[i]]=Dictogram([data[i+1]])
return markov_model
```

В реализации выше у нас есть словарь, который хранит окна в качестве ключа в паре «(ключ, значение)» и распределения в качестве значений в этой паре.

Структура цепи Маркова N-го порядка

```
from histograms import Dictogram


defmake_higher_order_markov_model(order, data):
    markov_model =dict()

for i inrange(0,len(data)-order):
        # Создаемокно
window=tuple(data[i: i+order])
        # Добавляемвсловарь
if window in markov_model:
# Присоединяем к уже существующему распределению
            markov_model[window].update([data[i+order]])
else:
            markov_model[window]=Dictogram([data[i+order]])
return markov_model
```

Очень похоже на цепь Маркова первого порядка, но в данном случае мы храним **кортеж** в качестве ключа в паре «(ключ, значение)» в словаре. Мы используем его вместо списка, так как кортеж защищен от любых изменений, а для нас это важно — ведь ключи в словаре меняться не должны.

Парсинг модели

Отлично, мы реализовали словарь. Но как теперь совершить генерацию контента, основываясь на текущем состоянии и шаге к следующему состоянию? Пройдемся по нашей модели:

```
from histograms import Dictogram
import random
from collections import deque
import re


defgenerate_random_start(model):
# Чтобы сгенерировать любое начальное слово, раскомментируйте строку:
# return random.choice(model.keys())

# Чтобы сгенерировать "правильное" начальное слово, используйте код ниже:
    # Правильные начальные слова - это те, что являлись началом предложений в
корпусе
if'END'in model:
        seed_word ='END'
while seed_word =='END':
            seed_word =model['END'].return_weighted_random_word()
return seed_word
return random.choice(model.keys())


defgenerate_random_sentence(length, markov_model):
    current_word =generate_random_start(markov_model)
sentence=[current_word]
for i inrange(0, length):
        current_dictogram = markov_model[current_word]
        random_weighted_word =
current_dictogram.return_weighted_random_word()
        current_word = random_weighted_word
sentence.append(current_word)
sentence[0]= sentence[0].capitalize()
return' '.join(sentence)+'.'
```

```
return sentence
```

Лучше генерировать «правильные» начальные слова — те, которые и в исходном тексте стояли в начале предложения. Для этого мы в словаре находим все ключи *«END»* и выбираем слово, следующее за одним из них. После генерации начального слова мы ищем, какое слово может идти дальше, обращаясь к тому же словарю, и выбираем нужное на основании комбинации вероятности и случайности. Продолжаем это делать, пока предложение не достигнет установленной нами длины, и в конце возвращаем его.

Контрольные вопросы

1. Что такое марковский процесс, марковская цепь?
2. Что такое инспекция программного кода?
3. Что такое парсинг модели?

Лабораторная работа № 21

Тема «Разработка приложений для моделирования процессов и явлений. Отладка приложения**»**

Цель работы: ознакомиться с инструментальными средствами моделирования деловых процессов, их классификацией и примерами.

Теоретическая часть:

Анализ параметров компьютера обычно выполняется с помощью достаточно универсальных утилит, таких, в частности, как Everest.

Включение в разрабатываемые приложения кода анализа оборудования, программного обеспечения и других подобных возможностей делает приложения более гибкими и, возможно, эффективными.

Определение версии операционной системы

Номер версии операционной системы и текущую платформуможно определить с помощью класса OSVersion. В следующихстроках кода эта информация выводится в поле компонента ListBox спомощью метода Items.Add.

Пример 1

```
Dim Pla As String = Environment.OSVersion.Platform.ToString()
Dim Ver As String = Environment.OSVersion.Version.ToString()
ListBox1.Items.Add("Платформа " + Pla)
                ListBox1.Items.Add("Версия " + Ver)
```

Получение системной даты и времени

Текущие дату и время можно получить с помощью операторов ***DateTime.Today*** и ***DateTime.Now***, первый из которых дает только дату, а второй дату и время.

Пример 2

```
ListBox1.Items.Add(DateTime.Now.ToString())
ListBox1.Items.Add(DateTime.Today.ToString())
```

Чтение реестра. Для обращения к реестру используются классы библиотекиMicrosoft.Win32. Оператор OpenSubKey открывает раздел реестра, изкоторого с помощью функции GetValue выбираются сведения о типепроцессора и его производителе.

Пример 3

```
Dim M As RegistryKey = Registry.LocalMachine
M = M.OpenSubKey( _
```

```
"HARDWARE\DESCRIPTION\System\CentralProcessor\0")
Dim Prc As Object = M.GetValue("Identifier")
Dim Vnd As Object = M.GetValue("VendorIdentifier")
ListBox1.Items.Add(Prc.ToString())
ListBox1.Items.Add(Vnd.ToString())
```

Из следующего раздела извлекаются сведения о статусе BIOS.

Пример 4

```
Dim V As RegistryKey = Registry.LocalMachine
V = V.OpenSubKey( _
"HARDWARE\DESCRIPTION\System\MultiFunctionAdapter\4")
Dim I_B As Object = V.GetValue("Identifier")
ListBox1.Items.Add(I_B.ToString())
```

Далее читаются дата записи BIOS и аппаратная платформакомпьютера.

Пример 5

```
RegistryKey B = Registry.LocalMachine
B = B.OpenSubKey("HARDWARE\\DESCRIPTION\\System\\")
Object D_B = B.GetValue("SystemBiosDate")
Object P_M = B.GetValue("Identifier")
```

Практическая часть

Информация о процессах. Для получения сведений о процессах в программе необходимассылка на пространство имен System.Diagnostics. Информацияизвлекается с помощью оператора Process.GetProcesses. Междуназванием процесса и его номером выводится символ табуляцииChr(9).

Пример 6

```
Dim Proc() As Process = Process.GetProcesses()
For Each P As Process In Proc
P.Refresh()
ListBox1.Items.Add(P.ProcessName+Chr(9)+P.Id.ToString())
Next
```

Список установленного программного обеспечения. Для чтения из реестра информации об установленномпрограммном обеспечении в программе требуется поместить ссылкуна пространство имен Microsoft.Win32. Для размещения массивастрок в поле компонента ListBox используется метод Items.AddRange.

Пример 7

```
Dim R As RegistryKey = Registry.LocalMachine.OpenSubKey( _
"Software\Microsoft\Windows\CurrentVersion\Uninstall")
Dim SubKeys() As String = R.GetSubKeyNames()
ListBox1.Items.AddRange(SubKeys)
```

Переменныеокружения. Для получения информации о переменных окружения впрограмме требуется поместить ссылку на пространство именSystem.Management. Информация о переменных окруженииявыбирается с помощью запроса Win32_Environment изинструментария управления Windows, затем отображается в полеListBox.

Пример 8

```
Dim Query As WqlObjectQuery = _
New WqlObjectQuery("Select * from Win32_Environment")
Dim Find As ManagementObjectSearcher = _
New ManagementObjectSearcher(Query)
For Each Mo As ManagementObject In Find.Get()
ListBox1.Items.Add(Mo("Description") + " - " + _
```

```
Mo("Name") + " - " + Mo("UserName") + " - " + _
Mo("VariableValue"))
Next
```

Контрольные вопросы

1. Что называется отладкой программы?
2. Что такое инспекция программного кода?
3. Назовите основной способ отладки?

Лабораторная работа № 22

Тема «Интеграция модуля в информационную систему**»**

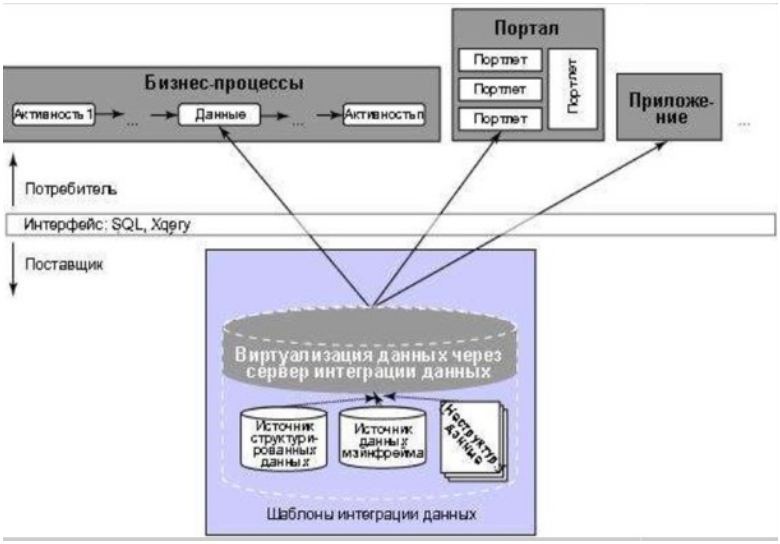
Цель работы: ознакомиться с инструментальными средствами моделирования деловых процессов, их классификацией и примерами.

Теоретическая часть

Интеграция—это не просто механическое объединение модулей информационной системы. При разработке плана интеграции исходят прежде всего из стратегических целей развития предприятия, возможного изменения бизнес-логики, в соответствии с которой выстраиваются бизнес-процессы и осуществляется их информационное сопровождение. Интеграция может производиться на уровне форматов и баз данных, программно-аппаратных и сетевых устройств, пользовательских интерфейсов, документооборота, программных приложений и т.д. форм и шаблонов

Интеграция на уровне данных

Одной из главных проблем интеграции данных является обилие форматов и типов (неструктурированные, частично-структурированные, жёстко-структурированные) данных, а также лавинообразное нарастание их объёмов. Циркулирование разнородных массивов данных и информации в сетях различных служб предприятия создает множество проблем с их сбором, структурированием, обработкой, анализом, хранением, архивированием и передачей пользователю для принятия делового решения.



Традиционная схема интеграции данных

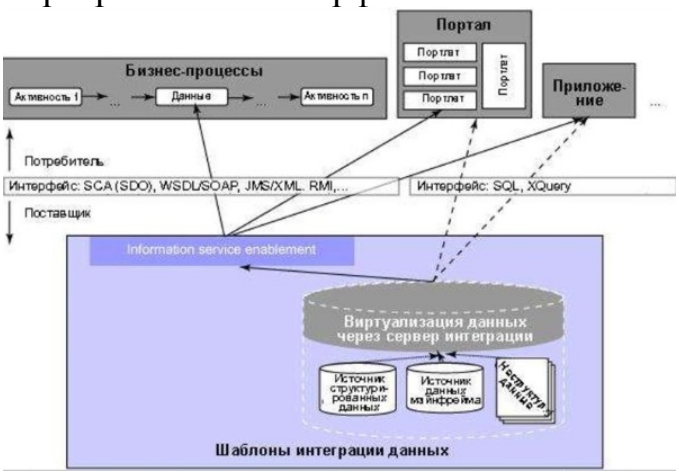
Интеграция на уровне физических, программных и пользовательских интерфейсов

Этот вид интеграции начинался как один из видов "лоскутной интеграции", когда предпринимались попытки объединить разрозненные программные приложения, написанные в разное время разными разработчиками, в подобие единого целого. Приложения объединялись по принципу "каждый с каждым", что, в конечном счёте, усложняло их взаимодействие и создавало массу проблем. Кроме того, всё сложнее становилось использовать унаследованные (Legacy Software) и встроенные (Embedded System) системы.

Такой подход хорош для небольшого количества приложений. При большом их числе он практически не работает и не позволяет строить качественно новые запросы к агрегированным данным, т.е. существенного выигрыша от объединения данных нет. В настоящее время проблема интеграции на уровне интерфейсов решается на базе использования информационных подсистем, реализованных стандартными программными приложениями с открытыми интерфейсами (Open Application Programming Interface).

Подобные унифицированные интерфейсы разрабатываются, например, на базе семейства международных стандартов POSIX. В этом случае степень интегрируемости можно характеризовать некоторым числовым показателем (метрикой) который можно, условно говоря, вычислить, перемножив показатель "качества" и "показатель открытости" программного интерфейса. Показателем качества могут выступать такие характеристики, как "совместимость", "надёжность", "переносимость", "понятность", "удобство использования" и пр. В результате мы получим индекс, который (в известной степени) характеризует способность приложения быть частью какого-то другого, глобального композитного приложения.

В настоящее время всё чаще применяется следующий алгоритм: отделяют слой обработки данных от привязанных к ним форм визуализации и реализуют прикладную бизнес-логику на одном из языков третьего поколения (3GL), оформив программный доступ к прикладным функциям в виде хорошо документированного программного интерфейса



Организация доступа к интегрированным данным через открытыеинтерфейсы

Интеграция на функционально-прикладном и организационном уровнях

Этот вид интеграции предполагает объединение ряда однотипных или схожих функций вмакрофункции с перераспределением потоков данных и управления, а также ресурсов имеханизмов для исполнения. Это часто влечёт за

собой перестройку организационныхструктур,бизнес-процессови,соответственно,схемуихинформационногоидокументационного обеспечения.

Выгоды от такой интеграции очевидны — процессы становятся более прозрачными,управляемыми, менее затратными, уменьшается количество обслуживающего персонала,число ошибок при формировании документов и т.д. Однако интеграция такого вида влечёт засобой существенную перестройку или полный реинжиниринг сети процессов, что связано скрупными рисками. Чаще всего такая интеграция проводится в том случае, когдапредприятие готовится к внедрению КИС на базе известного решения, которое требуетпривести бизнес-процессы к требуемому стандарту, или перестраивает свою деятельность всвязи со сменой устремлений, открытием филиалов в других странах, освоением новыхсегментов рынка и т.д.

Интеграция на уровне корпоративных программных приложений

Интеграция на уровне приложений (Enterprise Application Integration — EAI,) подразумеваетсовместное использование исполняемого кода, а не только внутренних данныхинтегрируемых приложений. Программы разбиваются на компоненты, которыеинтегрируются с помощью стандартизованных программных интерфейсов и специальногосвязующего ПО.

При таком подходе из этих компонентов создается универсальное программное ядро илиплатформа, с помощью которых используют все приложения. Для каждого приложениясоздается только один интерфейс для связи с этим ядром, что существенно облегчает задачуинтеграции. Полученную в результате систему легче поддерживать и расширять. Повторноеиспользование функций в рамках имеющейся среды позволяет значительно снизить время истоимость разработки приложений. В этом случае анализ внутренней конструкцииприложений — обязательный этап в оценке степени интегрируемости тех приложений,которые предполагается связывать в рамках того или иного проекта. Этот анализ усложняетсятем, что обычно разработчики приложений, являющихся законченными программнымипродуктами, как правило, не показывают деталей внутренней конструкции приложений.

В связи с этим технология интеграции в настоящее время рассматривает не просто интеграцию приложений, но их интеграцию на базе интеграции бизнес-процессов – в этом случае следуетговорить об интеграции на уровне всего предприятия (Enterprise Integration Metodology — EIM).

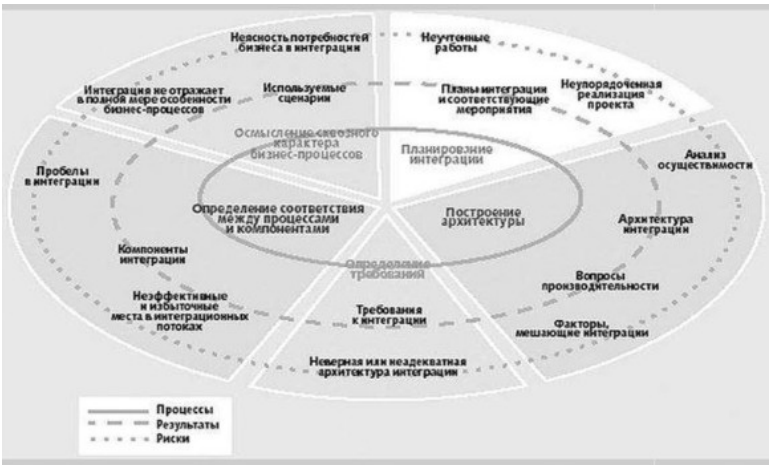


Схема применения методологии EIM

Интеграция при помощи Web-сервисов

Самый современный и быстро развивающийся подход к интеграции приложений. Он основанна обеспечении стандартного для Web-служб интерфейса доступа к приложениям и данным

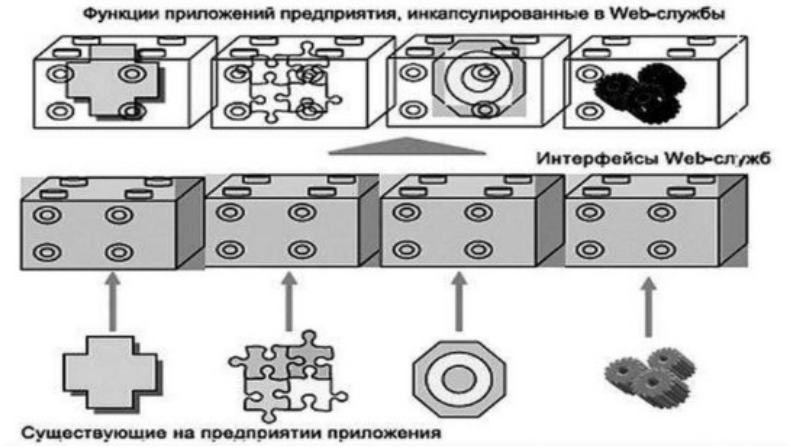


Схема доступа с использованием Web-служб

Возможность осуществлять оперативное управление распределенной компанией и ведениеконсолидированного управленческого учета по нескольким филиалам;

возможность осуществлять планомерное развитие общекорпоративной информационнойсистемы, интегрируя в нее функциональные компоненты, исходя из приоритетов развития бизнесакомпании и потребностей функциональных подразделений, т.е. возможность синхронизироватьразвитие системы с развитием бизнеса;

возможность при необходимости заменить любой функциональный компонент другим, болеесоответствующим текущим бизнес-потребностям;

возможность инвестировать в развитие информационных технологий не сразу, а поэтапно, накаждом этапе соотнося вложенные средства с полученным бизнес-эффектом, а также снижатьобщую стоимость автоматизированного рабочего места, включая затраты на создание системы,поддержку рабочих мест и обучение пользователей;

Резкое снижение времени сбора информации, необходимой для принятия управленческих иделовых решений, сокращение времени и трудозатрат на ведение учетных операций, наформирование промежуточных отчетов, на сверку

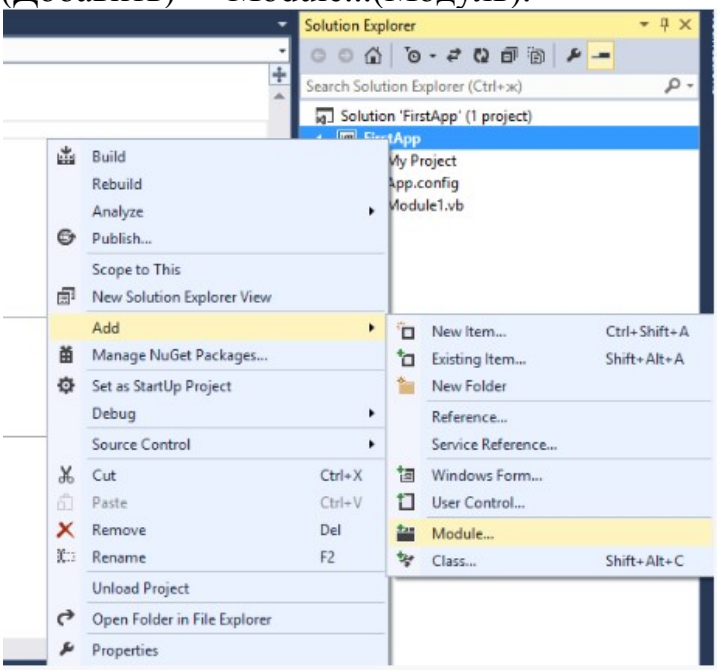
информации между подразделениями иликвидация противоречивости и несовместимости данных от различных служб;сохранение инвестиций в имеющиеся системы и оборудование, в обучение персонала.

Практическая часть

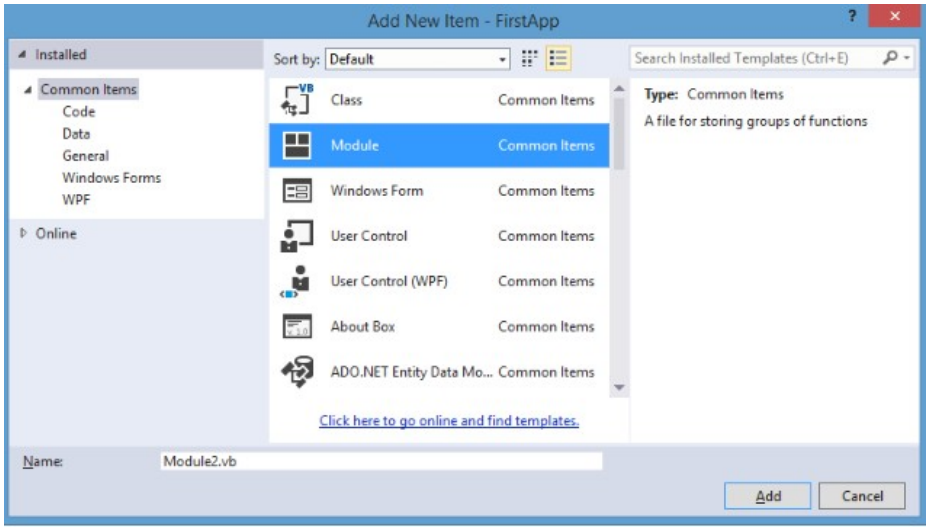
Поскольку Visual Basic.NET является полноценным объектно-ориентированным языком, для организации программного кода используются классы. Либо также могут использоваться модули. При создании нового консольного приложения Visual Studio автоматически генерирует следующий код:

```
Module Module1
    Sub Main()
    End Sub
EndModule
```

В данном случае наша программа представляет модуль с именем Module1. Концепция модулей представляет парадигму модульного программирования, согласно которому вся программа делится на ряд модулей, которые отвечают за разные функции программы. Пока мы использовали только один модуль в программе. Теперь создадим программу из двух модулей - первый модуль будет считывать из файла некоторое значение, а другой модуль будет получать это значение и проводить с ним некоторые операции. Чтобы добавить в программу второй модуль, нажмите справа в окне Solution Explorer (Обозреватель решений) на название проекта правой кнопкой мыши, затем в появившемся списке выберите пункт Add (Добавить) -> Module...(Модуль).



В открывшемся диалоговом окне выберите пункт Module (Модуль), оставьте в качестве его имени Module2 и нажмите кнопку Add (Добавить)



Таким образом, мы добавили в программу новый модуль Module2. В его коде ничего не определено, кроме объявления самого модуля:

```
Module Module2
End Module
```

Этот модуль будет отвечать у нас за считывание значения из файла. Чтобы считать файл, воспользуемся классом StreamReader, определенным в пространстве имен System.IO. Поэтому нам нужно импортировать данное пространство имен с помощью оператора Imports. Импортирование пространства имен производится в самом начале программы перед определением модуля или класса:

```
Imports System.IO
Module Module2
End Module
```

Что такое пространство имен? Пространства имен являются контейнерами для модулей, классов и других пространств имен. Одно и или несколько пространств имен и составляют приложения или библиотеки dll, построенные на платформе .NET. Мы можем и наш модуль поместить в пространство имен, которое назовем к примеру Modules. Этоделаетсяспомощьюключевогослова Namespace:

```
Imports System.IO
Namespace Modules
    Module Module2
End Module
End Namespace
```

Теперь перейдем к самой реализации нашей программы - определим функцию, которая будет в качестве параметра принимать путь к файлу и будет возвращать считанное значение:

```
Imports System.IO
Namespace Modules
    Module Module2
        Function Read(path As String) As Integer
'Число, которое возвращаем из функции
            Dim number As Integer
Try
                'Потокдлясчитывания
                Dim sr As New StreamReader(path)
'Считываем первый символ в файле
                number = Int32.Parse(sr.ReadLine())
'Закрываемпоток
sr.Close()
            Catch ex As Exception
```

```
Console.WriteLine(ex.Message)
            End Try
                        'Возвращаемрезультат
            Return number
        End Function
End Module
End Namespace
```

Обратите внимание на конструкцию Try ... Catch ... End Try - она нужна нам для обработки ошибок. Мы могли бы ее не использовать, но при выполнении программы может возникнуть ошибка. Например, мы введем неверный путь к файлу, и чтобы программа не зависла, а продолжала работать, мы используем данную конструкцию. После выражения Catch определен код для вывода ошибки на экран: Console.WriteLine(ex.Message).

Весь код нашей программы сосредоточен в трех строках между Try и Catch:

```
Dim sr As New StreamReader(path)
number = Int32.Parse(sr.ReadLine())
sr.Close()
```

В первой строке мы создаем поток для считывания файла, который мы получаем из параметра path. Чтобы создать новый объект используется ключевое слово New. Во второй строке мы считываем первый символ из файла. Метод ReadLine класса StreamReader считывает одну строку из файла, поэтому нам надо будет потом ее привести к типу Integer и полученное значение присвоить переменной number. В третьей строке мы закрываем поток методом Close.

Теперь перейдем к главному модулю. Он будет получать результат из модуля Module2 и вычислять факториал числа:

```
ModuleModule1
SubMain()
Console.Write("Введитепутькфайлу: ")
'Вводим с клавиатуры полный путь к файлу
Dim path As String = Console.ReadLine()
'Получаем число из файла, используя модуль Module2
Dim number As Integer = Modules.Module2.Read(path)
Console.WriteLine("Факториалчисла {0} равен {1}", number, Factorial(number))
Console.ReadLine()
    End Sub
    Function Factorial(x As Integer) As Integer
        If (x = 1) Then
            Return 1
        Else
            Return x * Factorial(x - 1)
        End If
    End Function
End Module
```

Сначала мы вводим путь к файлу, передаем его в функцию Read, которая определена в модуле Module2. Так как мы для модуля Module2 определили пространство имен, то надо указать и его, поэтому вызов метода имеет следующий вид:

```
Пространство_имен_модуля.Модуль.Метод_модуля
```

Хотя, если бы оба модуля находились в одном пространстве имен, то мы бы могли не указывать пространство имен. Итак, допустим, у нас на диске E находится файл Module.txt, в которое записано число 5. Тогда запустим программу, нажав F5, введем полный путь к файлу и получим факториал числа 5.

Контрольные вопросы

1. В чем заключается парадигма модульного программирования?
2. Что такое класс, модуль в VB.Net?
3. Методом закрывается поток данных?

Лабораторная работа № 23

Тема «Программирование обмена сообщениями между модулями**»**

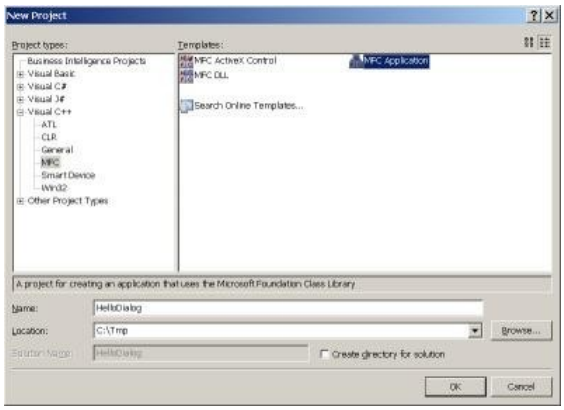
Цель работы: ознакомиться со способами программирования между сообщений модулями программы.

Теоретическая часть

Файлы к данной лабораторной работе, Вы можете скачать https://www.intuit.ru/studies/courses/594/450/lecture/10021?page=1

Практическая часть

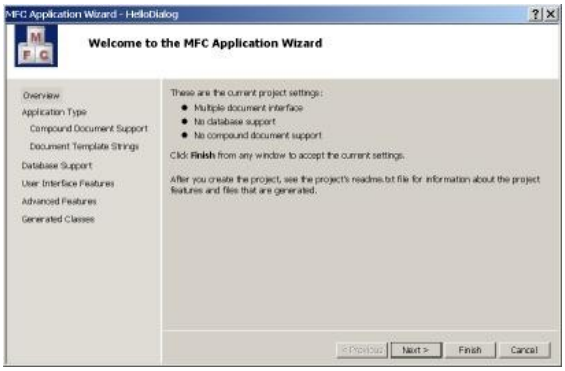
- Создайте командой File/New/Project новый проект с именем HelloDialog



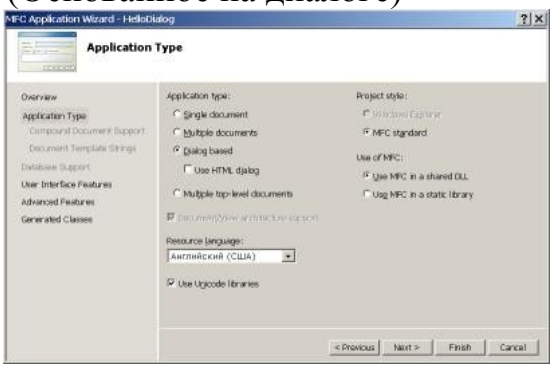
В диалоговом окне New Project нужно выбрать тип проекта MFC Application, в текстовом *поле* Name задать имя проекта (HelloDialog), в *поле* ввода Location или через кнопку Browse (Выбрать) задать *место* расположения папки с проектом. Каждый проект *оболочка*располагает в отдельной папке с именем проекта. Туда помещаются все вспомогательные файлы и файлы с кодом программы. Итоговый загрузочный (исполняемый) *модуль* программы (*файл* с расширением .exe) также будет носить имя проекта (в нашем случае HelloDialog.exe)

Мастер создания приложений MFC Application Wizard слева имеет *список* вкладок, позволяющих задать параметры проекта, а справа - сами вкладки с элементами управления проектом. В зависимости от типа создаваемого проекта те или иные опции на вкладках мастера будут заблокированы.

Первая вкладка *Overview* (Обзор) отображает параметры проекта, установленные в данный момент



● Откройте вкладку Application Type (Тип приложения) и включите радиокнопку Dialog based (Основанное на диалоге)



● Если в раскрывающемся списке Resource language нет языка **Русский**, то оставьте эту установку как есть. Позднее вручную мы настроим файл ресурсов на правильное восприятие приложением надписей на русском языке

● Остальные настройки вкладки Application Type оставьте в состоянии по умолчанию

● Откройте вкладку User Interface Features (Возможности пользовательского интерфейса) и заполните текстовое поле Dialog title содержимым заголовка **Диалоговое приложение MFC**

Возможно, что после создания проекта заголовок окна исказится и его нужно будет корректировать после настройки ресурсов на русский язык.

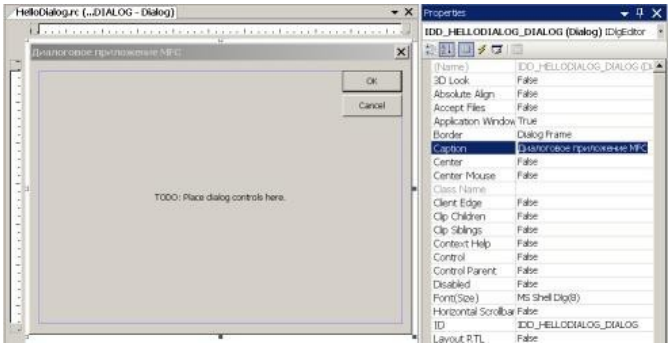
● Щелкните на кнопке Finish и мастер приложений создаст папку с именем проекта и запишет в нее все необходимые для проекта файлы

По правой и левой границам окна среды проектирования будут расположены пиктограммы выдвигающихся панелей, обеспечивающих удобный *интерфейс* для программиста. Любую панель можно включить через *меню* View, если ее нет на рабочем столе.

● Раскройте панель Resource View и настройте ресурсы на русский язык. Для этого раскройте все узлы дерева ресурсов и последовательно выделяя каждый узел в окне Properties из раскрывающегося списка Language выберите Русский

В заключение нужно подправить заголовок приложения, если он исказился при создании проекта. Для этого

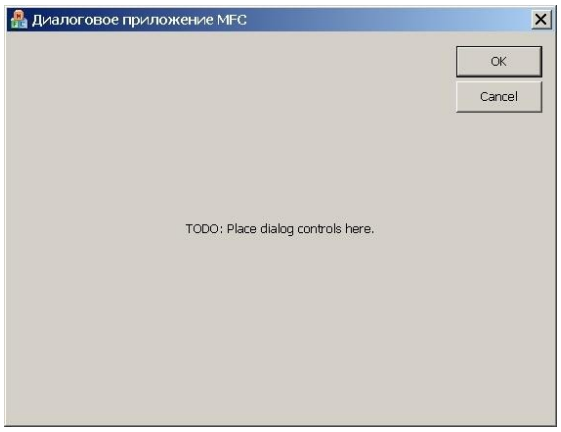
● Раскройте панель Resource View и двойным щелчком по узлу IDD_HELLODIALOG_DIALOG дерева ресурсов вызовите редактор Dialog Editor. Появится форма диалоговой панели



- Выделите форму и через панель свойств Properties установите в свойстве Caption требуемый заголовок

Построение каркаса приложения мастером MFC Application Wizard на этом закончено. Это работоспособное *приложение*, но оно пока имеет минимальную функциональность.

- Постройте приложение и убедитесь, что представление окна имеет вид



Программирование заготовки приложения

Создание программы включает в себя два этапа:

1. Визуальное проектирование
2. Написание кода

Подработаем наш проект и наполним его некоторой функциональностью, а именно:

1. Уберем статический текст на панели с дежурной надписью " TODO: Place dialog controls here "
2. Уменьшим диалоговую панель
3. Изменим надписи на кнопках на русские " **Выполнить** " и " **Отмена** "
4. Разместим в центре элемент управления Static Text и под ним элемент Edit Control из панели инструментов Toolbox
5. Сделаем так, чтобы после ввода текста в поле редактирования и нажатии на кнопке " **Выполнить** " появлялось диалоговое окно с введенным текстом

- Уберите дежурную надпись, щелкнув на ней и нажав клавишу Delete или в контекстном меню команду Delete
- Уменьшите диалоговую панель, выделив ее и потянув за маркер в нижнем правом углу
- Измените надписи на кнопках после их поочередного выделения и вызова панели свойств клавишей **F4**. В поле Captionпоменяйте значения "OK" и " Cancel " на русские " **Выполнить** " и " **Отмена** "

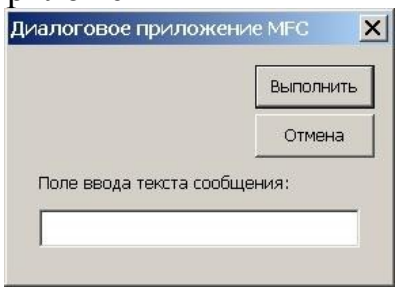
Вспомним, что во всех ресурсах приложения мы установили язык **Русский**.

● Из панели Toolbox поместите на форму текстовое поле Edit Control и текстовую метку Static Text

● Выровняйте элементы пользовательского интерфейса относительно панели и друг друга с помощью панели инструментов редактора диалоговых окон Dialog Editor. Он расположен в верхней части окна проектирования ниже системного меню и стандартной панели и имеет вид

Панель инструментов редактора диалоговых окон

● Щелкните на кнопке Test Dialog панели инструментов редактора диалоговых окон, чтобы получить визуальное представление пользовательской формы без компиляции всего приложения

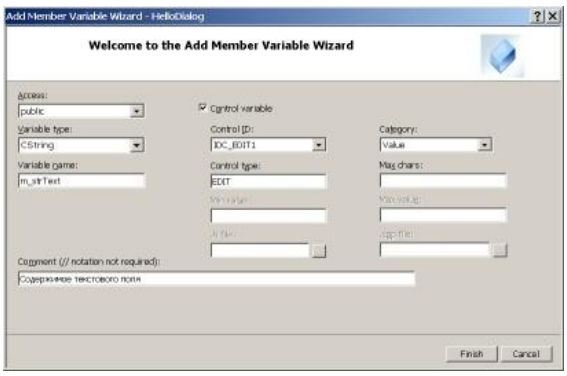


Теперь нужно связать *поле* ввода, кнопку " **Выполнить** " и библиотечное *диалоговое окно* сообщений AfxMessageBox() или MessageBox()между собой так, чтобы после ввода сообщения в *поле* ввода и нажатии кнопки " **Выполнить** " появлялось бы стандартное *диалоговое окно*сообщений с набранным текстом.

Введем вспомогательную переменную m_strText, в которой будем сохранять содержимое поля ввода и выводить его в стандартном диалоговом окне сообщений.

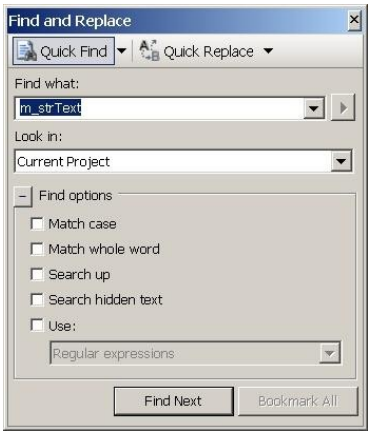
● Выделите правой кнопкой мыши текстовое поле Edit Control на шаблоне диалогового окна и через контекстное меню выполните пункт Add Variable, чтобы запустить мастер Add Member Variable Wizard

● Настройте мастер в соответствии с рисунком и щелкните на кнопке Finish



Для того, чтобы посмотреть, что сделал мастер при добавлении переменной m_strText, выполните следующее:

● Вызовите окно поиска и замены комбинацией клавиш Ctrl+F или командой Edit/Find and Replace/Quick Find меню оболочки и настройте его в соответствии с рисунком



Перемещаясь *по* ссылкам окна Find and Replace мы видим, что мастер переменных включил *описание переменной* в *класс*CHelloDialogDlg одноименного заголовочного файла

```
Объявление переменной в классе CHelloDialogDlg
class CHelloDialogDlg : public CDialog
{
  ..................................................
public:
  // Содержимое текстового поля
  CString m_strText;
};
```

В методе этого класса, отвечающего за *обмен данными* между элементами управления и переменными, он внес строку связи с текстовым полем

Функция обмена данными между текстовым полем и переменной

```
void CHelloDialogDlg::DoDataExchange(CDataExchange* pDX)
{
  CDialog::DoDataExchange(pDX);
  DDX_Text(pDX, IDC_EDIT1, m_strText);
}
```

В конструкторе класса мастер создал код инициализации переменной пустым значением

Инициализация переменной в конструкторе класса CHelloDialogDlg

```
CHelloDialogDlg::CHelloDialogDlg(CWnd* pParent /*=NULL*/)
  : CDialog(CHelloDialogDlg::IDD, pParent)
  , m_strText(_T(""))
{
  m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}
```

● Задайте вручную в конструкторе класса свой начальный текст для текстового поля

Инициализация переменной в конструкторе класса CHelloDialogDlg

```
CHelloDialogDlg::CHelloDialogDlg(CWnd* pParent /*=NULL*/)
  : CDialog(CHelloDialogDlg::IDD, pParent)
  , m_strText(_T("Элемент текстового поля"))
{
  m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}
```

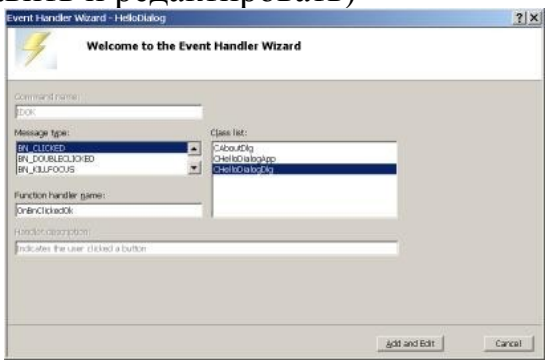
Теперь свяжем событие нажатия кнопки "**Выполнить**" с функцией обработки этого события, в которой передадим содержимое поля ввода,

сохраненное системой в промежуточной переменной, диалоговому окну сообщений.

Создание обработчика для дочерней кнопки

Все *элементы управления*, находящиеся на форме диалогового окна, являются его дочерними элементами. Они управляются этим родительским окном, наследуют от него стилевые свойства, а также получают от него сообщения операционной системы. Создадим для кнопки "**Выполнить**" обработчик для вызова стандартного диалогового окна сообщений.

- Вызовите контекстное меню для кнопки "**Выполнить**" на шаблоне диалогового окна и выполните команду Add *Event Handler*(Добавить обработчик событий), чтобы запустить мастер *Event Handler* Wizard
- Установите значения полей мастера как показано на рисунке и щелкните на кнопке Add and Edit (Добавить и редактировать)



Мастер создаст обработчик события в виде метода

Заготовка обработчика кнопки "Выполнить"

```
void CHelloDialogDlg::OnBnClickedOk()
{
  // TODO: Add your control notification handler code here
  OnOK();
}
```

Вызов метода OnOK() будет завершать работу диалогового окна. Одновременно **в карте сообщений** класса CHelloDialogDlg появится дополнительная строка, связывающая событие ON_BN_CLICKED о нажатии клавиши, имеющей *идентификатор* IDOK, с функцией-обработчиком OnBnClickedOk()

Карта сообщений класса CHelloDialogDlg с регистрацией обработчика

```
BEGIN_MESSAGE_MAP(CHelloDialogDlg, CDialog)
  ON_WM_SYSCOMMAND()
  ON_WM_PAINT()
  ON_WM_QUERYDRAGICON()
  //}}AFX_MSG_MAP
  ON_BN_CLICKED(IDOK, &CHelloDialogDlg::OnBnClickedOk)
END_MESSAGE_MAP()
```

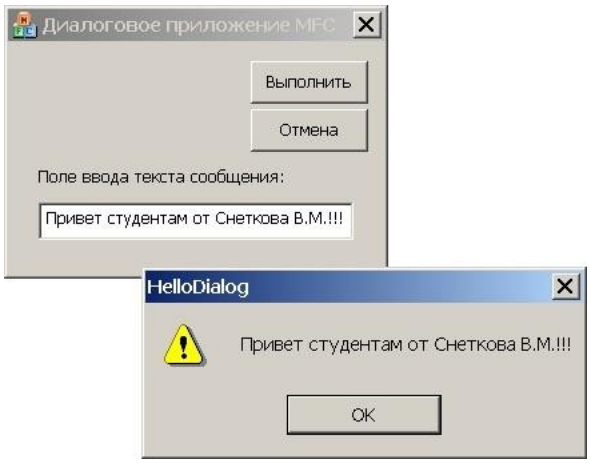
☞ Измените код обработчика OnBnClickedOk() на следующий

Код обработчика OnBnClickedOk()

```
void CHelloDialogDlg::OnBnClickedOk()
{
  UpdateData(TRUE);
  AfxMessageBox(m_strText);
}
```

В этом обработчике первой строкой кода мы инициируем выполнение функции DoDataExchange(), осуществляющей *копирование* значения текстового поля (и всех других текстовых полей ввода, если они есть) в промежуточную переменную m_strText, а во второй строке кода вызываем стандартное *диалоговое окно* сообщений для показа содержимого этой переменной.

- Постройте приложение и испытайте его функциональность на данном этапе



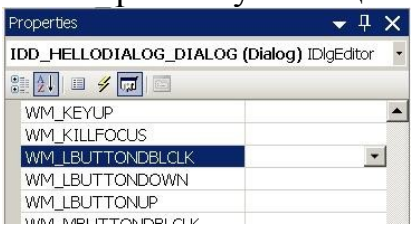
Создание обработчика для родительского окна

Теперь создадим обработчик сообщения для главного окна приложения, перехватывающего сообщения от самой операционной системы. Пусть, когда *пользователь* дважды щелкнет *по* клиентской области родительского окна самого приложения, должно появиться уведомление об этом. Здесь для доступа к таким сообщениям редактор дочерних элементов диалогового окна не поможет.

- Откройте панель Class View командой View/Class View
- Найдите класс CHelloDialogDlg, отвечающий за главное диалоговое окно, выделите его и раскройте вкладку Properties(Свойства)
- На вкладке Properties вверху щелкните по пиктограмме Messages (Сообщения)

В окне появятся два столбца с перечнем всех сообщений, для которых можно создать обработчики, и перечень имен всех уже созданных обработчиков.

- Найдите и выделите сообщение WM_LBUTTONDBLCLK. Внизу окна появится краткий комментарий к выбранному сообщению.



- Добавьте через раскрывающийся список обработчик этого сообщения Среда сама создаст функцию-член CHelloDialogDlg::OnLButtonDblClk() со стандартным именем и добавит элемент карты сообщений, переадресующий сообщение WM_LBUTTONDBLCLK на вызов этой функции. Карта сообщений станет выглядеть так

Карта сообщений класса CHelloDialogDlg после регистрации обработчика

```
BEGIN_MESSAGE_MAP(CHelloDialogDlg, CDialog)
  ON_WM_SYSCOMMAND()
```

```
  ON_WM_PAINT()
  ON_WM_QUERYDRAGICON()
  //}}AFX_MSG_MAP
  ON_BN_CLICKED(IDOK, &CHelloDialogDlg::OnBnClickedOk)
  ON_WM_LBUTTONDBLCLK()
END_MESSAGE_MAP()
```

Обратите внимание, что среда не позволяет регистрировать обработчики родительского диалогового окна с произвольными именами, а назначает им стандартное имя. *По* этой причине в карте сообщений нет необходимости связывать для события конкретные элементы с конкретными обработчиками, как это требуется делать для дочерних элементов.

Объявление функции-обработчика помещается в заголовочный *файл* диалогового окна и имеет вид

Объявление обработчика в заголовочном файле HelloDialogDlg.h

```
class CHelloDialogDlg : public CDialog
{
  ..............................................
public:
  // Содержимое текстового поля
  CString m_strText;
public:
  afx_msg void OnBnClickedOk();
public:
  afx_msg void OnLButtonDblClk(UINT nFlags, CPoint point);
};
```

Заготовка функции-обработчика щелчка *по* диалоговому окну, которую автоматически добавит среда, имеет вид

Обработчик двойного щелчка на родительском диалоговом окне приложения

```
void CHelloDialogDlg::OnLButtonDblClk(UINT nFlags, CPoint point)
{
  // TODO: Add your message handler code here and/or call default

  CDialog::OnLButtonDblClk(nFlags, point);
}
```

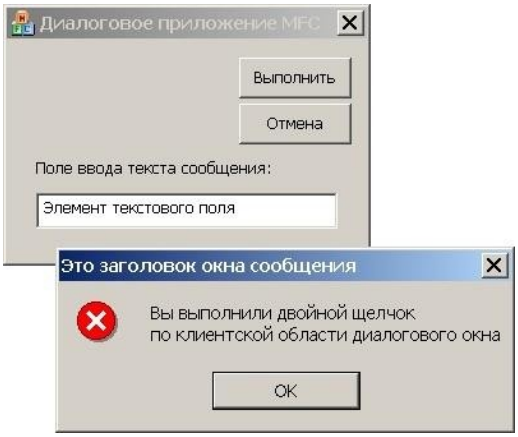
● Скорректируйте заготовку обработчика события родительского диалогового окна так

Обработчик двойного щелчка на родительском диалоговом окне приложения

```
void CHelloDialogDlg::OnLButtonDblClk(UINT nFlags, CPoint point)
{
  // TODO: Add your message handler code here and/or call default
  CString mess;
  mess =  "Вы выполнили двойной щелчок \n"
      "по клиентской области диалогового окна";
  CString title;
  title = "Это заголовок окна сообщения";
  MessageBox(mess, title, MB_ICONSTOP);

  CDialog::OnLButtonDblClk(nFlags, point);
}
```

● Постройте приложение, которое после двойного щелчка по клиентской области будет иметь вид



Итак, для каждого добавленного обработчика сообщений среда генерирует код в трех места:

1. Объявление в заголовочном файле .h
2. Определение в файле .cpp
3. Внесение записи в карту сообщений файла .cpp

Постройте приложение и проверьте его работоспособность

Контрольные вопросы

1. Что в проекте MFC Application отвечает за создание приложения, основанного на диалоге?
2. Что такое инспекция программного кода?
3. Как вызвать форму диалоговой панели?

Лабораторная работа № 24

Тема «Организация файлового ввода-вывода данных**»**

Цель работы: получение практических навыков в работе с файловыми функциями форматного ввода-вывода.

Теоретическая часть

Составить программу, которая читает текстовый файл и выполняет такое преобразование его, какое задано в Вашем варианте индивидуального задания.

Примечания:

1. Файл должен иметь не менее 10-15 строк текста.
2. Допускается ограничить максимальную длину строки в тексте 80 символами.
3. Допускается (если в индивидуальном задании не оговорено второе) считать, что слова разделяются пробелами, а знаки препинания рассматривать как буквы.
4. Допускается создавать при выполнении программы временные файлы, которые, однако, не должны сохранятся после окончания программы.

Разработка алгоритма решения.

Общий алгоритм

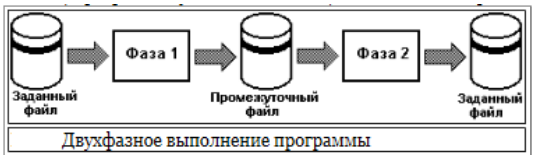
Для решения задачи придется прочитать файл, как минимум, дважды - при первом чтении (первая фаза программы) определить максимальную длину строки, а при втором (вторая фаза программы) - выполнять заданные преобразования. В строке могут быть лишние пробелы, которые при выравнивании будут удалены, так что простого определения длины (например, при помощи функции ***strlen()***) недостаточно - следует при определении длины принять во внимание и пробелы, что будет требовать посимвольного просмотра всей строки. Если мы в первой фазе будем только подсчитывать лишние пробелы, то во второй фазе нам в значительной степени придется повторить эту работу, чтоб избавиться от этих лишних пробелов - так почему бы не выполнить всю эту работу в первой фазе? Выравнивание во второй фазе будет выполняться добавлением пробелов между словами, так что нам нужно будет знать количество слов В каждой строке - его можно определить в том же просмотре строки, когда мы удаляем лишние пробелы.

Таким образом, мы приходим к двухфазной схеме программы, когда в первой фазе выполняется какая-то часть обработки и получается промежуточный результат. Этот промежуточный результат является входными данными для второй фазы, которая выполняет окончательную обработку и формирует конечный результат.

Промежуточный результат первой фазы состоит из:

- максимальной длины строки в файле;
- всех строк файла, из которых удалены лишние пробелы;
- количества слов для каждого строки файла.

Если первая составная часть промежуточного результата - одно число, которое может сохраняться в переменной программы и через нее передаваться от первой фазы во вторую, то остальные составляющие требуют для своего сохранения файл. Программа будет выполняться, как показано на рис.



Двухфазное выполнение программы

Общий алгоритм выполнения программы (без детализации выполнения первой и второй фаз) показан на рис.

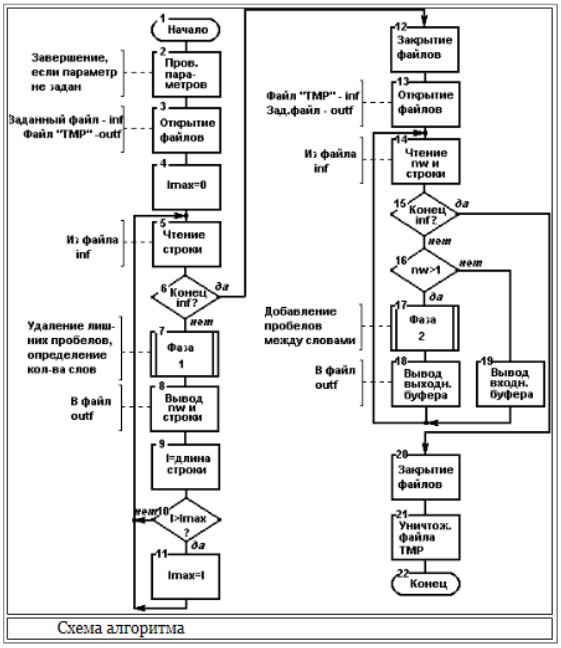


Схема алгоритма

На схеме приняты такие обозначения для переменных алгоритма: ***inf*** и ***outf*** - файловые переменные, которые представляют входной и выходной файлы соответственно; ***l*** - длина текущей строки, ***lmax*** - длина самой длинной строки в файле; ***nw*** - количество слов в текущей строке.

Программа должна получать имя файла как свой параметр. Поэтому выполнение программы должно начинаться (блок 1) с проверки того, задан ли этот параметр. Если параметр не задан, программа должна завершаться.

Открытие файлов (блоки 3 и 13) должно сопровождаться проверкой успешности открытия. Если открытие неуспешно (причиной этого может быть, например, неправильно заданное имя файла), программа должна завершаться.

Та обработка строк, которую выполняют две фазы программы на рис.2, не детализирована - подробно мы раскроем эти фазы ниже.

При первом чтении файла выполняется определение максимальной длины строки - на схеме алгоритма это блоки 4 и 9 - 11.

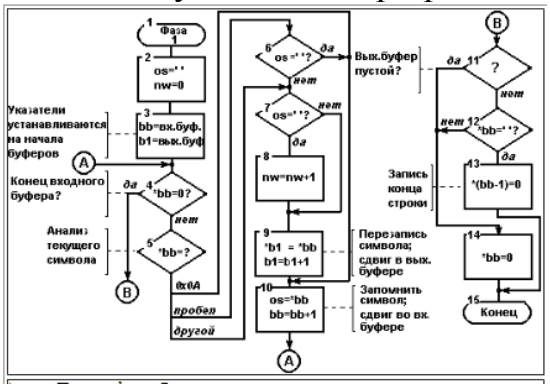
Алгоритм второго чтения (блоки 13 - 20) предусматривает некоторые обстоятельства, которые мы ранее не рассмотрели. Выравнивание будет вестись вставкой дополнительных пробелов между словами. Но что делать, если строка состоит из единственного слова или вообще пустая? Задание не предусматривает ничего по этому поводу - принимаем решение, что такая строка будет выводиться без выравнивания, только с удалением лишних пробелов. Обработка строки на второй фазе будет, очевидно, происходить таким образом, что прочитанная строка будет сохраняться в каком-то входном буфере, а строка-результат будет формироваться в выходном буфере. Следовательно, если количество слов в строке больше одного (блок 16), выполняется обработка фазы 2 и в выходной файл выводится строка из выходного буфера (блоки 17, 18). Если же в строке

одно слово или совсем ничего нет, строка из входного буфера без обработки выводится в выходной файл (блок 19).

Обратите внимание также на то, что перед завершением программы временный файл для сохранения промежуточного результата уничтожается (блок 21).

Алгоритм выполнения первой фазы

Детализируем теперь ту обработку каждого строки, которая выполняется на первой фазе, - рисунок 3. Имея в виду то, что реализация алгоритма буде выполнена на языке С, мы уже при составлении его учитываем возможности языка в отношении использования указателей при работе со строками.



Первая фаза. Схема алгоритма.

Та обработка, которую мы должны выполнить, включает в себя удаление двойных пробелов и подсчет количества слов. Для обеих задач нам нужно сравнивать два соседних символа в строке. К текущему символу мы будем обращаться через указатель, а предыдущий символ будет сохраняться в переменной ***os***. Для самого первого символа строки значение предыдущего символа - пробел, начальное значение количества слов - ***nw*** - 0 (блок 2). Строка из файла считывается в буфер в оперативной памяти, при ее посимвольном просмотре мы будем формировать выходную строку в втором буфере. Указатели ***b1*** и ***b2*** устанавливаются на начало входного и выходного буферов соответственно (блок 3).

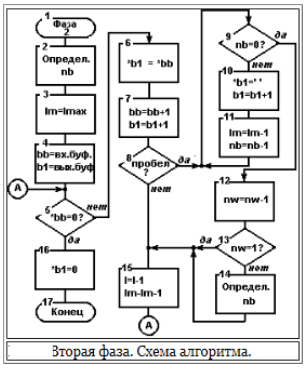
Обработка строки будет вестись в цикле (блоки 4 - 10). Условием выхода из цикла является достижение конца строки во входном буфере, это выяснится, когда мы найдем во входном буфере символ с кодом 0 (блок 4). В каждой итерации цикла мы первым делом проверяем текущий символ во входном буфере - тот, на который указывает указатель ***b1*** (блок 5). Если это символ перехода на новую строку - символ с кодом $\boldsymbol{0A_{16}}$ - мы не делаем ничего, сразу же переходим на конец цикла (блок 10). Если это пробел, проверяем (блок 6), не был ли пробелом предыдущий символ, если так, то переходим на конец цикла. Таким образом, повторные пробели и символы перехода просто не копируются в выходной буфер. Если пробел не повторный, он обрабатывается как любой второй символ. Обработка других символов начинается с проверки, не был ли пробелом предыдущий символ (блок 7). Такая ситуация является признаком начала нового слова и, если она обнаружена, счетчик слов увеличивается на 1 (блок 8). Потом символ из входного буфера пересылается в выходной и указатель в выходном буфере сдвигается на 1 (блок 9). На следующем блоке сходятся все ветви обработки: тот символ входного буфера, который был только что обработан,

сохраняется как предыдущий символ и сдвигается указатель во входном буфере (блок 10).

После выхода из цикла нам нужно записать в выходной буфер признак конца строки - символ с кодом 0. Но нужно иметь в виду то, что мы можем иметь еще один лишний пробел в конце строки. Если выходная строка пустая (блок 11) или последний записанный в нее символ - не пробел (блок 12), признак конца записывается (блок 14) за последним символом строки (туда, куда показывает указатель ***b2***). Если же последний символ - пробел, признак конца записывается вместо него (блок 13).

Алгоритм выполнения второй фазы

Цель этой фазы - выравнивание. Если обозначим ту длину строки, до которой должно происходить выравнивание, - ***lmax***, а текущую длину - ***l***, то для выравнивания в строку следует добавить ***lmax-l*** пробелов. Эти пробелы должны быть равномерно распределены в ***nw-1*** промежутках между словами. Следовательно, количество пробелов, которое следует добавлять в каждый промежуток - ***nb=(lmax-l)/(nw-1)*** . Но операция деления может давать результат с дробной частью. Следовательно, для точного выравнивания в несколько промежутков в начале строки будут добавляться ***nb*** пробелов (и игнорированием дробной части ***nb)*** , а в остальные промежутки - на 1 большее количество, так чтобы итоговая длина строки была равна точно ***lmax***. Количество последних промежутков равно остатку от деления. Другой способ точного определения требуемого количества пробелов для каждого промежутка - пересчитывать ***nb*** для каждого следующего промежутка с учетом количества уже обработанных символов строки, именно такой способ предлагается в алгоритме, схема которого приведена на рисунке



Вторая фаза. Схема алгоритма.

nb определяется (блок 2), как было показано выше. Выше мы также приняли такой вариант определения точного количества пробелов, который предусматривает переопределение ***nb***. При этом переопределении нужно будет корректировать значения ***l*** и ***lmax***, поэтому делаем копию значения ***lmax*** в переменной ***lm*** (блок 3), именно эту копию мы и будем изменять. Потом устанавливаем указатели ***b1*** и ***b2*** на начало соответствующих буферов (блок 4) и в цикле (блоки 5 - 15) перебираем символы входного буфера, доки не встретим признак конца строки (блок 5).

В каждой итерации цикла мы переписываем один символ из входного буфера в выходной (блок 6) и сдвигаем указатели в буферах (блок 7). Потом проверяем - не был ли только что переписанный символ пробелом (блок 8). Если так, то в цикле (блоки 9 - 11) записываем в выходной буфер ***nb*** пробелов. После обработки каждого промежутка уменьшаем счетчик слов (блок 12) и, если еще есть

промежутки (блок 13), перевычисляем ***nb*** (блок 14). После выхода из цикла в выходной буфер дописывается еще символ конца строки (блок 16).

Обратите внимание на то, что те переменные, которые используются для вычисления ***nb***, уменьшаются на 1 при обработке каждого следующего символа: при обработке символа из входного буфера уменьшаются и ***l***, и ***lm*** (блок 15), а при дописывании каждого пробела в выходной буфер - только ***lm*** (блок 11).

Определение переменных программы

Символьная строка, в которой сохраняется имя файла, который мы обрабатываем: `char filename[80];`

Два буфера для сохранения входной и выходной строк, одни те же буферы используются на первой и на второй фазе. Размер обоих буферов - 81 символ (80 символов - ограничение, которое допускается условиями задания, + 1 символ - признак конца строки).

```
char bu1[81], bu2[81];
```

Символьные строки, как данные большого объема, должны размещаться в статической памяти. Остальные переменные - локальные в функции ***main()***.

Для реализации алгоритма нам прежде всего будут нужны файловые переменные - для входного и выходного файлов. Мы будем пользоваться функциями форматного ввода-вывода, которые требуют объявления файлов як переменных типа ***FILE****:

FILE *inf, *outf;

Эти две программные переменные используются и в первой, и во второй фазе программы, хотя в разных фазах они ассоциируются с разными физическими файлами.

Работа с символьными строками будет вестись через указатели, следовательно, нужны будут указатели на текущие символы во входной и выходной строке соответственно:

char *b1, *b2;

В соответствии с алгоритмом нам нужна переменная для хранения предыдущего символа:

char os;

Переменные для хранения: текущей длины строки, максимальной длины строки, максимальной длины строки с модификацией ее (см. алгоритм 2-ой фазы):

int l; int lmax; int lm;

Переменные для хранения: количества слов и количества дополнительных пробелов между словами:

int nw; int nb;

Всем целочисленным переменным мы даем тип ***int*** из соображений стилевой традиции, хотя объективно хватило бы и типа ***short***.

Разработка текста программы

Файлы, которые включаются в программу:

- описания функций форматного ввода-вывода;
- описания строковых функций (***strlen()***,***string()***);
- описания функций общего назначения (***exit()***,***atoi()***).

Функция ***main()*** имеет два параметра, это форматный состав параметров, которые передаются главной функции: второй параметр - массив указателей на

символьные строки, первый - количество элементов в этом массиве. Элемент с индексом 0 - строка обращения к программе, а тот параметр, которого требуют спецификации (имя файла) - элемент с индексом 1. Проверка параметров состоит в проверке их количества.

Для открытия файлов повсюду в программе используется функция ***fopen()***, она возвращает указатель, который является файловой переменной в программе. Мы всегда проверяем этот указатель; если он пустой, мы выдаем на экран сообщение про невозможность открытия файла и завершаем программу с помощью функции ***exit()***.

Заголовок цикла:

```
for (lmax=0; fgets(bu1,80,inf)!=NULL; ) {
```

реализует сразу блоки 4, 5, 6 схемы алгоритма рис.2. Для чтения строки мы применяем функцию ***fgets()***, которая считывает строку из файла. При попытке читать за концом файла эта функция возвращает ***NULL***, что и есть признаком конца файла.

Следующий сложный заголовок цикла:

```
for (os=' ',nw=0,b1=bu1, b2=bu2; *b1; os=*b1++) {
```

реализует блоки 2,3,4,5,10 схемы алгоритма рис.3.

Оператор:

```
switch (*b1) { . . . }
```

- разветвление в зависимости от значения текущего символа. Обратите внимание на то, как реализована та часть разветвления, которая на схеме рис.3 представлена блоком 6, в программе это:

```
case ' ': if (os==' ') break;
```

- если предыдущий символ - пробел, происходит выход из ***switch***, если же нет - управление переходит наследующий оператор, т.е., на обработку остальных символов.

Сложное условие исключения пробелов в конце строки (блоки 11 - 14 рис.3) в программе реализовано одним оператором ***if - else***:

```
if ((b2>=bu2)&&(*(b2-1)==' ')) *(b2-1)=0;
else *b2=0;
```

Вывод промежуточного результата выполняется оператором:

```
fprintf(outf,"%02d %s\n",nw,bu2);
```

Этот оператор выводит текстовую строку, в которой два первые символы - цифры количества слов в строке, далее через пробел - сама строка.

Цикл чтения строк на второй фазе:

```
while (fgets(bu1,80,inf)!=NULL) {
```

Необходимо пояснить разделение прочитанной строки на количество слов и саму строку. Мы записываем признак конца строки в элемент с индексом 2:

```
bu1[2]=0;
```

таким образом мы разбиваем строку на две строки. Первая из этих строк - два первых символа, мы преобразуем их в число слов:

```
nw=atoi(bu1);
```

Указатель ***b1*** мы устанавливаем на элемент с индексом 3, т.е., на начало самих данных строки:

```
b1=bu1+3;
```

Потом мы еще определяем длину строки и записываем признак конца вместо последнего символа строки - символа перехода на новую строку:

```
l=strlen(b1)-1; b1[l]=0;
```

Поскольку в формуле вычисления ***nb*** мы используем ***nw-1***, мы сразу уменьшаем ***nw*** на 1 и далее сравниваем его з 0:

```
if (--nw>0) {
```

Сложный заголовок цикла:

```
for (lm=lmax, b2=bu2; *b1; l--,lm--) {
```

реализует блоки 3,4,5,15 схемы алгоритма рис.4.

Оператор:

```
if ((*b2++=*b1++)==' ') {
```

реализует блоки 6, 7, 8 схемы алгоритма рис.4: пересылка, наращивание указателей, анализ пересланного символа.

Вывод результатов в файл выполняется функцией ***fprintf()***, но для обработанной строки параметр функции - ***b2*** - указатель на выходной буфер, а для необработанной - на входной - ***b1***.

Уничтожение временного файла выполняется функцией ***unlink()***.

Полный текст программы приведен ниже.

```
/*****************************************************/
/* Лабораторная работа №24 */
/* Обработка текстового файла */
/* Пример выполнения. Вариант №30. */
/*****************************************************/
#include
#include
#include
charfilename[80]; /* имяфайла */
char bu1[81], bu2[81]; /* входной и выходной буферы */
int main(int an, char *av[]) {
char *b1, *b2; /* текущие указатели в буферах */
char os; /* предыдущий символ */
FILE *inf, *outf; /* файловые переменные */
int l, /* текущая длина строки */
lmax, /* максимальная длина строки */
lm; /* макс. длина строки (рабочая) */
int nw, /* количество слов */
nb; /* количество пробелов */
/* проверка параметров */
if (an<2) {
printf("Не задан параметр вызова\n");
exit(0);
}
strcpy(filename,av[1]);
/* открытие файлов */
if ((inf=fopen(filename,"r"))==NULL) {
printf("Невозможно открыть файл: %s\n",filename);
exit(0);
}
if ((outf=fopen("TMP","w"))==NULL) {
printf("Невозможно создать файл:TMP\n");
exit(0);
}
/* первая фаза - определение максимальной длины */
```

```
/* чтение фала строка за строкой */
for (lmax=0; fgets(bu1,80,inf)!=NULL; ) {
/* удаление из строки лишних пробелов
и определение количества слов */
for (os=' ',nw=0,b1=bu1, b2=bu2; *b1; os=*b1++) {
switch(*b1) {
case 0xA: /* символ новой строки удаляется */
break;
case ' ': /*2-й пробел подряд удаляется */
if (os==' ') break;
default: /* остальные символы переписываются */
if (os==' ') nw++; /* начало слова */
*b2++=*b1;
break;
}
}
/* удаление пробела в конце */
if ((b2>=bu2)&&(*(b2-1)==' ')) *(b2-1)=0;
else *b2=0;
/* вывод в файл количества слов и строки */
fprintf(outf,"%02d %s\n",nw,bu2);
/* определение максимальной длины */
l=strlen(bu2);
if (l>lmax)lmax=l;
}
fclose(inf);
fclose(outf);
/* вторая фаза - окончательная обработка */
/* открытиефайлов */
if ((inf=fopen("TMP","r"))==NULL) {
printf("Невозможно открыть файл:TMP\n");
exit(0);
}
if ((outf=fopen(filename,"w"))==NULL) {
printf("Невозможно открыть файл: %s\n",filename);
exit(0);
}
/* чтение фала строка за строкой */
while (fgets(bu1,80,inf)!=NULL) {
/* виделение количества слов и текста */
bu1[2]=0; nw=atoi(bu1);
b1=bu1+3;
l=strlen(b1)-1; b1[l]=0;
/* если слов 1 или 0 - строка остается как есть */
if (--nw>0) {
/* определение количества пробелов между словами */
nb=(lmax-l)/nw;
for (lm=lmax, b2=bu2; *b1; l--,lm--) {
/* перезапись символов на выход */
if ((*b2++=*b1++)==' ') {
/* если конец слова - добавить пробели */
for (; nb; nb--) { lm--; *b2++=' ';}
/* коррекция количества пробелов между словами */
if (--nw) nb=(lm-l)/nw;
}
}
/* признак конца строки в выходном буфере */
*b2=0;
/* выводвфайл */
fprintf(outf,"%s\n",bu2);
```

```
}
/* вывод строки, которая осталась без изменений */
else fprintf(outf,"%s\n",b1);
}
fclose(inf);
fclose(outf);
/* уничтожениепромежуточногофайла */
unlink("TMP");
return 0;
}
```

Отладка программы

Для отладки программы прежде всего следует подготовить текст, который будет использоваться как входные данные контрольного примера. В этом тексте должны быть такие строки, которые позволяют проверить разные ветви алгоритма программы, а именно:

- строки, в которых между некоторыми словами есть промежутки в два и более пробелов;
- строки, в которых есть пробелы в начале строки;
- строки, в которых есть пробелы в конце строки;
- строки, которые состоят из одного слова;
- строки, которые состоят из одного слова с пробелами перед ним и после него;
- строки, которые состоят из одних пробелов;
- пустые строки.

Обязательно следует проверить также работу программу при отсутствии входных параметров и при задании имени несуществующего файла.

Поскольку программа изменяет файл, рекомендуем на этапе отладки записывать результат не в той же файл, а в другой. Тогда подготовленный файл с данными контрольного примера можно буде использовать много раз. Рекомендуем также при отладке удалить из программы оператор уничтожения промежуточного файла - это дает возможность контролировать промежуточные результаты.

Отладку самого преобразования строк можно вести в пошаговом режиме с отслеживанием содержимого входного и выходного буферов и значений ключевых переменных.

Практическая часть

Вариант №1

В каждой строке удалить лишние пробелы между словами и, сохраняя первоначальную длину строки, разместить текст по центру строки.

Вариант №2

В каждой строке заменить последовательность слов "один", "два", "три" на "1-2-3".

Вариант №3

В каждой строке вставить после знаков препинания пробели, если их там нет.

Вариант №4

В каждой строке поменять местами соседние слова.

Вариант №5
Разбить каждую строку на две строки приблизительно одного размера, не разрывая слова.
Вариант №6
В каждой строке все слова, длина которых превышает среднюю длину слов в строке, сократить до средней длины.
Вариант №7
Разместить текст в несколько столбцов, так чтоб n-е слово i-ой строки размещалось под n-им словом i+1-ой строки.
Вариант №8
Скопировать второе слово каждой строки в начало следующей строки.
Вариант №9
В каждой строке удалить пары слов, в которых одно слово является зеркальным отображением второго.
Вариант №10
В каждой строке удалить лишние пробелы между словами и разместить их по левому краю, сохраняя первоначальную длину строки.
Вариант №11
Удалить из текста все слова, состоящие из одной буквы.
Вариант №12
Удалить из текста все слова с четными номерами (сквозная нумерация слов по всему тексту).
Вариант №13
В каждом слове все буквы разместить в алфавитном порядке.
Вариант №14
В каждой строке удалить те слова, в которых первые три буквы совпадают с начальными буквами последнего слова строки.
Вариант №15
В каждом слове строки изменить порядок букв на противоположный.
Вариант №16
Во всем тексте вставить переход на новую строку там, где есть два или больше пробела подряд.
Вариант №17
В каждой строке поменять местами первое слово с последним, второе - с предпоследним и т.д.
Вариант №18
В каждой строке самое длинное слово заменить на "длинное слово".
Вариант №19
Удалить из текста все слова, в которых содержатся буквосочетания "ов".
Вариант №20
В каждой строке удалить лишние пробелы между словами и разместить их по правому краю, сохраняя первоначальную длину строки.
Вариант №21
В каждой строке заменить последовательность слов "1", "2", "3" на "один-два-три".

Вариант №22
Удалить из текста все слова, в которых какие-либо буквы повторяются.
Вариант №23
В каждой строке все слова разместить в алфавитном порядке.
Вариант №24
Переформировать весь текст так, чтоб каждое предложение занимало отдельную строку. (Признак конца предложения - точка.)
Вариант №25
В каждой строке все слова, которые совпадают с первым словом строки, заменить на второе слово.

Контрольные вопросы

1. Что называется отладкой программы?
2. Что такое инспекция программного кода?
3. Назовите основной способ отладки?

Лабораторная работа № 25

Тема «Разработка модулей экспертной системы**»**

Цель работы: освоение технологии и методики построения экспертных систем на примере разработанной учебной экспертной системы. Студент выступает в роли одновременно эксперта и инженера по знаниям.

Теоретическая часть

Экспертные системы - это прикладные системы ИИ, в которых база знаний представляет собой формализованные эмпирические знания высококвалифицированных специалистов (экспертов) в какой либо узкой предметной области. Экспертные системы предназначены для замены при решении задач экспертов в силу их недостаточного количества, недостаточной оперативности в решении задачи или в опасных (вредных) для них условиях.

Обычно экспертные системы рассматриваются с точки зрения их применения в двух аспектах: для решения каких задач они могут быть использованы и в какой области деятельности. Эти два аспекта накладывают свой отпечаток на архитектуру разрабатываемой экспертной системы.

Можно выделить следующие основные классы задач, решаемых экспертными системами:

- диагностика,
- прогнозирование,
- идентификация,
- управление,
- проектирование,
- мониторинг.

Наиболее широко встречающиеся области деятельности, где используются экспертные системы:

- медицина,
- вычислительная техника,

- военное дело,
- микроэлектроника,
- радиоэлектроника,
- юриспруденция,
- экономика,
- экология,
- геология (поиск полезных ископаемых),
- математика.

На рисунке изображена обобщенная структура экспертной системы.

База знаний предназначена для хранения экспертных знаний о предметной области, используемых при решении задач экспертной системой.

Структура экспертных систем

База данных предназначена для временного хранения фактов или гипотез, являющихся промежуточными решениями или результатом общения системы с внешней средой, в качестве которой обычно выступает человек, ведущий диалог с экспертной системой.

Машина логического вывода - механизм рассуждений, оперирующий знаниями и данными с целью получения новых данных из знаний и других данных, имеющихся в рабочей памяти. Для этого обычно используется программно реализованный механизм дедуктивного логического вывода (какая-либо его разновидность) или механизм поиска решения в сети фреймов или семантической сети.

Машина логического вывода может реализовывать рассуждения в виде:

1. дедуктивного вывода (прямого, обратного, смешанного);
2. нечеткого вывода;
3. вероятностного вывода;
4. унификации (подобно тому, как это реализовано в Прологе);
5. поиска решения с разбиением на последовательность подзадач;
6. поиска решения с использованием стратегии разбиения пространства поиска с учетом уровней абстрагирования решения или понятий, с ними связанных;
7. монотонного или немонотонного рассуждения,
8. рассуждений с использованием механизма аргументации;
9. ассоциативного поиска с использованием нейронных сетей;

10. вывода с использованием механизма лингвистической переменной.

Подсистема общения служит для ведения диалога с пользователем, в ходе которого ЭС запрашивает у пользователя необходимые факты для процесса рассуждения, а также, дающая возможность пользователю в какой-то степени контролировать и корректировать ход рассуждений экспертной системы.

Подсистема объяснений необходима для того, чтобы дать возможность пользователю контролировать ход рассуждений и, может быть, учиться у экспертной системы. Если нет этой подсистемы, экспертная система выглядит для пользователя как "вещь в себе", решениям которой можно либо верить либо нет. Нормальный пользователь выбирает последнее, и такая ЭС не имеет перспектив для использования.

Подсистема приобретения знаний служит для корректировки и пополнения базы знаний. В простейшем случае это - интеллектуальный редактор базы знаний, в более сложных экспертных системах - средства для извлечения знаний из баз данных, неструктурированного текста, графической информации и т.д.

Когда целесообразно использование экспертных систем

Экспертные системы целесообразно использовать тогда, когда 1) *разработка ЭС возможна*, 2) *оправдана* и 3) *методы инженерии знаний соответствуют решаемой задаче*.

Рассмотрим более подробно эти условия.

Разработка ЭС возможна, когда:

- существуют эксперты в данной области;
- эксперты должны сходиться в оценке предлагаемого решения;
- эксперты должны уметь выразить на естественном языке и объяснить используемые методы;
- задача требует только рассуждений, а не действий;
- задача не должна быть слишком трудной, ее решение должно занимать у эксперта до нескольких часов или дней, а не недель или месяцев;
- задача должна относиться к достаточно структурированной области;
- решение не должно использовать в значительной мере здравый смысл (т.е. широкий спектр общих сведений о мире и о способе его функционирования).

Разработка ЭС оправдана, если::

- решение задачи принесет значительный эффект:
- использовать человека-эксперта невозможно из-за ограниченного количества экспертов или из-за необходимости выполнения экспертизы одновременно во многих местах;
- при передаче информации эксперту происходит значительная потеря времени или информации;
- необходимо решать задачу в окружении, враждебном человеку.

Методы инженерии знаний соответствуют задаче, если задача обладает следующими характеристиками::

● может быть естественным образом решена посредством манипуляции с символами, а не с числами;

● имеет эвристическую природу, т.е. не годится задача, которая может быть решена гарантированно с помощью некоторых формальных процедур;

● должна быть достаточно сложной, чтобы оправдать затраты, но не чрезмерно сложной;

● должна быть достаточно узкой, но практически значимой.

Построить базу знаний по теме

1. Медицинские диагнозы и болезни
2. Электронные компоненты
3. Кулинарные рецепты
4. Ландшафтный дизайн
5. Зап.части к автомобилям

Практическая часть

В соответствии с вариантом, студентам предлагается изучить

1. Теоретический материал, презентации и модуля учебной экспертной системы, разработанным в среде ***EsWin***, реализующим мини-опрос по представленному материалу в соответствии с вариантом из раздела «Материалы к лабораторным работам – Вариант».

2. Инструментальное ПО для построения экспертных систем. В инструментальное ПО входят помимо экспертной оболочки ***EsWin*** программа-редактор баз знаний ***EDKB*** и программа для просмотра баз знаний ***KBVIEW***.

3. , реализованного в среде оболочки ***EsWin***.

После изучения теоретических и практических материалов предлагается выполнить:

4. Тестирование оболочки.
5. Провести анализ ошибок.
6. Доработать базу знаний
7. Выполнить реализацию и тестирование новой версии базы знаний.

После чего оформить отчет по выполненным лабораторным работам, в котором отразить:

● Цель разработки, предложенного в вашем варианте модуля учебной экспертной системы

● Результаты анализа выявленных ошибок

● Структуру фреймов, правил-продукций для исходного и доработанного модулей учебной экспертной системы

Контрольные вопросы

1. Что называется экспертной системой?
2. Какие задачи решают экспертные системы?
3. Какова структура экспертных систем?

Лабораторная работа № 26

Тема «Создание сетевого сервера и сетевого клиента**»**

Цель работы: ознакомиться с инструментальными средствами моделирования деловых процессов, их классификацией и примерами.

Теоретическая часть

Задача заключается в передаче зашифрованного сообщения по прослушиваемой сети. Две стороны заранее договариваются о ключе шифрования методом Цезаря. На стороне клиента сообщение должно шифроваться и передаваться по сети серверу, на стороне сервера должно происходить дешифрование сообщения и вывод его на экран. Интерфейс работы с программой на стороне клиента:

```
 >>> Enter your message:
rrr
Enter the key number (1-26)
3
Send ciphertext: uuu
 >>>
```

Пользователь вводит сообщение, которое хочет передать по сети, указывает ключ шифрования. Программа выводит на экран зашифрованное сообщение и отправляет его серверу. Интерфейс работы с программой на стороне сервера:

```
>>>
Connected client
Enter the key number (1-26)
3
Received ciphertext: uuu
Plaintext: rrr
>>>
```

На экране пользователя со стороны сервера в первую очередь выводится сообщение о том, что установлена связь с клиентом. Далее указывается ключ шифрования и выводятся зашифрованный и открытый тексты

Работа с сокетами Применяемая в IP-сетях архитектура клиент-сервер использует IP-пакеты для коммуникации между клиентом и сервером. Клиент отправляет запрос серверу, на который тот отвечает. В случае с TCP/IP между клиентом и сервером устанавливается соединение (обычно с двусторонней передачей данных), а в случае с UDP/IP - клиент и сервер обмениваются пакетами (дейтаграммами) с негарантированной доставкой. Каждый сетевой интерфейс IP-сети имеет уникальный в этой сети адрес (IPадрес). Упрощенно можно считать, что каждый компьютер в сети Интернет имеет собственный IP-адрес. При этом в рамках одного сетевого интерфейса может быть несколько сетевых портов. Для установления сетевого соединения приложение клиента должно выбрать свободный порт и установить соединение с серверным приложением, которое слушает (listen) порт с определенным номером на удаленном сетевом интерфейсе. Пара IPадрес и порт характеризуют сокет (гнездо) - начальную (конечную) точку сетевой коммуникации. Для создания соединения TCP/IP необходимо два сокета: один на локальной машине, а другой - на удаленной. Таким образом, каждое сетевое соединение имеет IP-адрес и порт на локальной машине, а также IP-адрес и порт на удаленной машине. Модуль socket в Python обеспечивает доступ к интерфейсу BSD сокетов и обеспечивает возможность работать с сокетами из

Python. Сокеты используют транспортный уровень согласно семиуровневой модели OSI (Open Systems Interconnection, взаимодействие открытых систем).

Уровни модели OSI Физический: поток битов, передаваемых по физической линии. Определяет параметры физической линии. Канальный (Ethernet, PPP, ATM и т.п.): кодирует и декодирует данные в виде потока битов, справляясь с ошибками, возникающими на физическом уровне в пределах физически единой сети. Сетевой (IP): маршрутизирует информационные пакеты от узла к узлу. Транспортный (TCP, UDP и т.п.): обеспечивает прозрачную передачу данных между двумя точками соединения. Сеансовый: управляет сеансом соединения между участниками сети. Начинает, координирует и завершает соединения. Представления: обеспечивает независимость данных от формы их представления путем преобразования форматов. На этом уровне может выполняться прозрачное (с точки зрения вышележащего уровня) шифрование и дешифрование данных. Приложений (HTTP, FTP, SMTP, NNTP, POP3, IMAP и т.д.): поддерживает конкретные сетевые приложения. Протокол зависит от типа сервиса.

Каждый сокет относится к одному из коммуникационных доменов. Модуль socket поддерживает домены UNIX и Internet. Каждый домен подразумевает свое семейство протоколов и адресацию. Данное изложение будет затрагивать только домен Internet, а именно протоколы TCP/IP и UDP/IP, поэтому для указания коммуникационного домена при создании сокета будет указываться константа socket.AF_INET (и socket.SOCK_STREAM для надежной потокоориентированной службы). Далее представлены примеры двух программ, использующих протокол TCP/IP для передачи текстовой строки: сервер принимает строку и пересылает ее обратно клиенту.

Практическая часть

Исходный текст программы-сервера

```
 # Echo server program
import socket
HOST = '127.0.0.1'
PORT = 50007
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((HOST, PORT))
s.listen(1)
conn, addr = s.accept()
 print 'Connected client'
while 1:
data = conn.recv(1024)
if not data:
break
else:
print 'Received[2]: ', data
conn.send(data)
print 'Send[3]: ', data
conn.close()
```

Исходный текст программы-клиента

```
# Echo client program
import socket
HOST = '127.0.0.1'
PORT = 50007
```

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((HOST, PORT))
data = 'Hello world'
s.send(data)
print 'Send[1]: ', data
data = s.recv(1024)
s.close()
print 'Received[4]: ', data
```

Прежде всего, нужно запустить сервер, затем, запустив еще одну копию IDLE, запустить в ней клиент. Сервер открывает сокет на локальной машине на порту 50007, и адресе 127.0.0.1 (метод bind связывает локальный сетевой адрес транспортного уровня с сокетом). После этого сервер слушает (listen) порт. Когда на порту появляются данные, принимается (accept) входящее соединение, создается сокет, соответствующий новому соединению клиента и сервера. Сокет, для которого был вызван accept, остается в состоянии listen и готов к принятию следующих соединений. Метод accept возвращает пару – socket-объект и адрес удаленного компьютера, устанавливающего соединение (пара – IP-адрес, порт на удаленной машине). После этого можно применять методы recv и send для общения с клиентом. В recv задается число байтов в очередной порции, от клиента может прийти и меньшее количество данных. Код программы-клиента достаточно очевиден. Метод connec устанавливает соединение с удаленным хостом (в приведенном примере он расположен на той же машине). Данные передаются методом send и принимаются методом recv – аналогично тому, что происходит на сервере. Для реализации поставленной задачи можно воспользоваться модулем шифрования по методу Цезаря: импортировать модуль и воспользоваться необходимыми методами.

Контрольные вопросы

1. Что использует архитектура клиент-сервер для коммуникации между клиентом и сервером?
2. Что должно сделать приложение клиента для установления сетевого соединения?
3. Опишите уровни модели OSI?

Варианты заданий

1. Опишите процесс учета посещения студентов учебных занятий и успеваемости студентов с точки зрения работника деканата.

Разработать программный модуль «Учет успеваемости студентов». Программный модуль предназначен для оперативного учета успеваемости студентов в сессию деканом, заместителями декана и сотрудниками деканата. Сведения об успеваемости студентов должны храниться в течение всего срока их обучения и использоваться при составлении справок о прослушанных курсах и приложений к диплому.

2. Опишите процесс учета студентов, обучающихся в институте от процесса зачисления студента до получения диплома с точки зрения работника деканата.

Разработать программный модуль «Личные дела студентов». Программный модуль предназначен для получения сведений о студентах сотрудниками деканата, профкома и отдела кадров. Сведения должны храниться в течение всего срока обучения студентов и использоваться при составлении справок и отчетов.

3. Опишите процесс организации рабочего дня руководителя с точки зрения его секретаря.

Разработать приложение «Органайзер». Приложение предназначено для записи, хранения и поиска адресов и телефонов физических лиц и организаций, а также расписания, встреч и др. Приложение предназначено для организации рабочего дня руководителя.

4. Опишите процесс работы кафедры вуза с точки зрения преподавателя.

Разработать программный модуль «Кафедра», содержащий сведения о сотрудниках кафедры (ФИО, должность, ученая степень, дисциплины, нагрузка, общественная работа, совместительство и др.). Модуль предназначен для использования сотрудниками отдела кадров и деканата.

5. Опишите процесс работы лаборатории с точки зрения ее служащего.

Разработать программный модуль «Лаборатория», содержащий сведения о сотрудниках лаборатории (ФИО, пол, возраст, семейное положение, наличие детей, должность, ученая степень). Модуль предназначен для использования сотрудниками профкома и отдела кадров.

6. Опишите процесс работы химчистки с точки зрения ее служащего.

Разработать программный модуль «Химчистка». При записи на обслуживание заполняется заявка, в которой указываются ФИО владельца, описание изделия, вид услуги, дата приема заказа и стоимость услуги. После выполнения работ распечатывается квитанция.

7. Опишите процесс организации работы с нарушителями правил дорожного движения с точки зрения работника милиции.

Разработать программный модуль «Учет нарушений правил дорожного движения». Для каждой автомашины (и ее владельца) в базе хранится список нарушений. Для каждого нарушения фиксируется дата, время, вид нарушения и размер штрафа. При оплате всех штрафов машина удаляется из базы.

8. Опишите процесс работы автомагазина с точки зрения его служащего.

Разработать программный модуль «Картотека автомагазина», предназначенный для использования работниками магазина. В базе содержатся сведения об

автомобилях (марка, объем двигателя, дата выпуска и др.). При поступлении заявки на покупку производится поиск подходящего варианта. Если такого нет, клиент заносится в клиентскую базу и оповещается, когда вариант появляется.

9. Опишите процесс работы АТС с точки зрения ее служащего.

Разработать программный модуль «Картотека абонентов АТС». Картотека содержит сведения о телефонах и их владельцах. Фиксирует задолженности по оплате (абонентской и повременной). Считается, что повременная оплата местных телефонных разговоров уже введена.

10. Опишите процесс организации работы автостанции с точки зрения ее служащего.

Разработать программный модуль «Автокасса», содержащий сведения о наличии свободных мест на автобусные маршруты. В базе должны содержаться сведения о номере рейса, маршруте, водителе, типе автобуса, дате и времени отправления, а также стоимости билетов. При поступлении заявки на билеты программа производит поиск подходящего рейса.

11. Опишите процесс работы книжного магазина с точки зрения его служащего.

Разработать программный модуль «Книжный магазин», содержащий сведения о книгах (автор, название, издательство, год издания, цена). Покупатель оформляет заявку на нужные ему книги, если таковых нет, он заносится в базу и оповещается, когда нужные книги поступают в магазин.

12. Опишите процесс работы автостоянки с точки зрения ее служащего.

Разработать программный модуль «Автостоянка». В программе содержится информация о марке автомобиля, его владельце, дате и времени въезда, стоимости стоянки, скидках, задолженности по оплате и др.

13. Опишите процесс организации работы гостиницы с точки зрения администратора.

Разработать программный модуль «Гостиница», содержащий сведения о наличии свободных мест и о проживающих в гостинице. Программный модуль предназначен для бронирования мест в гостинице и оформления проживающих.

14. Опишите процесс организации работы детективного агентства с точки зрения ее работников.

Разработать программный модуль «Детективное агентство», содержащий сведения о клиентах агентства и об оказанных услугах. Программный модуль предназначен для учета средств за оказанные услуги.

15. Опишите процесс работы музея с точки зрения его служащего.

Разработать программный модуль «Музей», предназначенный для использования работниками музея. В базе содержатся сведения об экспонатах музея и вносятся данные при поступлении новых экземпляров. При выполнении инвентаризации данные заносятся в базу, проводится сверка и выдаются отчеты по учету экспонатов в музее.