Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования
«Амурский государственный университет»

Кафедра информационных и управляющих систем

УЧЕБНО-МЕТОДИЧЕСКИЙ КОМПЛЕКС ДИСЦИПЛИНЫ

«ТЕХНОЛОГИЯ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ»

Основной образовательной программы по направлению подготовки

230100.68 – Информатика и вычислительная техника

Благовещенск 2012

УМКД разработан канд. техн. наук, доцентом Т.А. Галаган

Рассмотрен и рекомендован на заседании кафедры

Протокол заседания кафедры от «__» _____201_ г. № _____

Зав. кафедрой _____ / А.В. Бушманов /
(подпись)

УТВЕРЖДЕН:

Протокол заседания УМСС 230100.68 - Информатика и вычислительная техника

от «____» _____ 201__ г. № _____

Председатель УМСС _____ / В.В. Еремина /
(подпись)

РАБОЧАЯ ПРОГРАММА

1. ЦЕЛИ И ЗАДАЧИ ДИСЦИПЛИНЫ

Целью освоения дисциплины «Технология разработки программного обеспечения» является изучение этапов проектирования, разработки и испытаний больших программных систем с точки зрения требований разработчика.

Задача дисциплины – обобщение знания, полученные студентами, и обеспечивает изучение современных технологий разработки программного обеспечения.

2. МЕСТО ДИСЦИПЛИНЫ В СТРУКТУРЕ ООП ВПО

Код ООП М.2.Б.2. – Профессиональный цикл, базовая (общепрофессиональная) часть.

Для изучения дисциплины «Технология разработки программного обеспечения» студент должен обладать навыками создания программ на языке высокого уровня, в том числе на основе объектно-ориентированного подхода, уметь анализировать и обобщать информацию; в объеме основной образовательной программы данного направления, работать с современным программным обеспечением.

Знания, полученные в результате изучение дисциплины «Технология разработки программного обеспечения» могут использоваться при работе над выпускной квалификационной работой.

3. КОМПЕТЕНЦИИ ОБУЧАЮЩЕГОСЯ, ФОРМИРУЕМЫЕ В РЕЗУЛЬТАТЕ ОСВОЕНИЯ ДИСЦИПЛИНЫ (МОДУЛЯ)

В результате освоения дисциплины обучающийся должен демонстрировать следующие результаты образования:

Знать

жизненный цикл программ, оценку качества программных продуктов, технологии разработки программных комплексов, CASE-средства;

Уметь использовать типовые программные продукты, ориентированные на решение научных, проектных и технологических задач;

Владеть методиками сбора, переработки и представления научно-технических материалов по результатам исследований к публикации и печати, а также в виде обзоров, рефератов, отчетов, докладов и лекций.

Обучение студентов данной дисциплине должно способствовать развитию следующих профессиональных компетенций:

способен проявлять инициативу, в том числе в ситуациях риска, брать на себя всю полноту ответственности (ОК-5);

разрабатывать и реализовывать планы информатизации предприятий и их подразделений на основе Web- и CALSтехнологий. (ПК-3),

формировать техническое задание и участвовать в разработке аппаратных или/и программных средств информационных и автоматизированных систем.(ПК-7)

4. СТРУКТУРА И СОДЕРЖАНИЕ ДИСЦИПЛИНЫ (МОДУЛЯ)

Общая трудоемкость дисциплины составляет 5 зачетных единицы, 180 часов.

№ п/п	Раздел дисциплины	Семестр	Неделя семестра	Формы текущего контроля успеваемости (по неделям семестра)			Форма промежуточной аттестации (по семестрам)
				прак	лаб.	сам.	
1	Этапы разработки про-	3	1 – 4	4	2	30	Отчеты по лаб.

	граммного обеспечения						работам
2	Методы разработки программного обеспечения	3	5 – 10	6	8	40	Отчеты по лаб. работам
3	Проектирование программного обеспечения на основе объектно-ориентированного подхода	3	11 – 12	2	4	36	Отчеты по лаб. работам Тест №1
4	Документация по сопровождению программных средств	3	13 – 15	3	4	30	Отчет по лаб. работам
5	Международные стандарты разработки программного обеспечения	3	16 – 18	3		8	Тест №2
	ИТОГО			18	18	144	180

5. СОДЕРЖАНИЕ РАЗДЕЛОВ И ТЕМ ДИСЦИПЛИНЫ

5.1 Практические занятия

5.1.1. Основные определения: программные средства, программное обеспечение, программный продукт. Классификация типов программного обеспечения. Понятие жизненного цикла программного обеспечения. Модели жизненного цикла программного обеспечения: каскадная, спиральная, инкрементальная. Этапы разработки программного обеспечения.

5.1.2. Методы разработки программного обеспечения. Проект. Состав и структура коллектива разработчиков, их функции. Нисходящий анализ процесса управления проектированием программного изделия. Восходящее проектирование. Метод последовательной модернизации. Блок-схемы, ER-диаграммы, UML-диаграммы, DFD-диаграммы,

5.1.3. Проектирование программного обеспечения на основе объектно-ориентированного подхода. Конструирование программных систем как структурных коллекций, реализующих абстрактные типы данных.

5.1.4. Документация по сопровождению программных средств. Стандарты и практические руководства. Организация выпуска документации в фазах исследования и анализа осуществимости. Организация выпуска документации в фазах конструирования и программирования. Организация выпуска документации в фазах оценки и использования.

5.1.5. Международные стандарты проектирования, разработки, оформления документации, пользовательского интерфейса.

5.2. Лабораторные работы

5.2.1. Методы написания программного обеспечения

5.2.2. Доказательство правильности программ

5.2.3. Технология написания тестов

6. САМОСТОЯТЕЛЬНАЯ РАБОТА

№ п/п	№ раздела (темы) дисциплины	Форма (вид) самостоятельной работы	Трудоемкость в часах
1	Этапы разработки программного обеспечения	Изучение учебной литературы Подготовка отчетов по лабораторным работам	30
2	Методы разработки программного обеспечения	Изучение учебной литературы Подготовка отчетов по лабораторным работам	40
3	Проектирование про-	Изучение учебной литературы	36

	граммного обеспечения на основе объектно-ориентированного подхода	Подготовка отчетов по лабораторным работам Подготовка к тестированию	
4	Документация по сопровождению программных средств	Изучение учебной литературы Подготовка отчетов по лабораторным работам	30
5	Международные стандарты разработки программного обеспечения	Изучение учебной литературы Подготовка отчетов по лабораторным работам Подготовка к тестированию	8
	Итого		144

7. МАТРИЦА КОМПЕТЕНЦИЙ УЧЕБНОЙ ДИСЦИПЛИНЫ

Разделы	Компетенции			Общее число компетенций
	ПК-3	ПК-7	ОК-5	
1	+	+	+	3
2	+	+	+	3
3	+	+	+	3
4	+	+	+	3
5	+	+	+	3

8. ОБРАЗОВАТЕЛЬНЫЕ ТЕХНОЛОГИИ

К образовательным технологиям, используемым в преподавании данной дисциплины, относятся практические и лабораторные работы.

В изложении материала на практических заданиях наряду используются такие неимитационные методы обучения, как:

проблемное занятия, начинается с постановки проблемы, которую необходимо решить в ходе изложения материала,

занятие с заранее запланированными ошибками, которые студенты должны обнаружить самостоятельно по мере изложения материала.

На занятиях используются компьютерные презентации. Удельный вес занятий, проводимых в интерактивных формах должен составлять не менее 20% аудиторных занятий.

Лабораторные работы проводятся в компьютерных классах и предназначены для решения прикладных задач с использованием современных инструментальных средств.

При проведении лабораторных работ используются неигровые имитационные методы обучения:

контекстное обучение, направленное на решение профессиональных задач,

работа в команде – совместная деятельность студентов в группе, направленная на решение общей задачи с разделением ответственности и полномочий.

При оценивании результатов обучения используется балльно-рейтинговая технология.

№ п/п	№ раздела (темы) дисциплины	Форма (вид) образовательных технологий	Кол-во часов
1	Этапы разработки программного обеспечения	Проблемное занятие	2
2	Методы разработки программного обеспечения	Мультимедийная презентация Проблемное занятие	2 2
3	Проектирование программного обеспечения на основе объектно-	Мультимедийная лекция	2

	ориентированного подхода		
4	Документация по сопровождению программных средств	Мультимедийная презентация Занятие с ошибками	3
5	Международные стандарты разработки программного обеспечения	Мультимедийная презентация	3

9. ОЦЕНОЧНЫЕ СРЕДСТВА ДЛЯ ТЕКУЩЕГО КОНТРОЛЯ УСПЕВАЕМОСТИ, ПРОМЕЖУТОЧНОЙ АТТЕСТАЦИИ ПО ИТОГАМ ОСВОЕНИЯ ДИСЦИПЛИНЫ И УЧЕБНО-МЕТОДИЧЕСКОЕ ОБЕСПЕЧЕНИЕ САМОСТОЯТЕЛЬНОЙ РАБОТЫ СТУДЕНТОВ

Для оценки текущей успеваемости в данной дисциплине относятся:
тестовые задания с закрытыми и открытыми видами вопросов;
отчеты по выполнению лабораторных работ;
зачет.

Вопросы к зачету

1. Программные средства и программное обеспечение
2. Классификация типов программного обеспечения.
3. Понятие жизненного цикла программного обеспечения.
4. Модели жизненного цикла программного обеспечения: каскадная, спиральная, инкрементальная.
5. Этапы разработки программного обеспечения.
6. Состав и структура коллектива разработчиков, их функции.
7. Организация интерфейса между модулями, написанными разными программистами.
8. Выполнение проекта
9. Методы программирования
10. Нисходящий анализ процесса управления проектированием программного изделия.
11. Восходящее проектирование.
11. Метод последовательной модернизации.
12. Структурное проектирование
13. Проектирование программного обеспечения на основе объектно-ориентированного подхода.
14. Конструирование программных систем как структурных коллекций, реализующих абстрактные типы данных.
15. Стратегия тестирования
16 Практические руководства.
17. Организация выпуска документации в фазах исследования и анализа осуществимости.
18. Организация выпуска документации в фазах конструирования и программирования.
19. Организация выпуска документации в фазах оценки и использования.
20 Стандарты проектирования программного обеспечения.
21. Стандарты оформления проектной документации.
22. Стандарты пользовательского интерфейса.

Примеры тестовых заданий

1. Какой стандарт устанавливает правила оформления экранов (шрифты и цветовая

палитра), состав и расположение окон и элементов управления; правила использования клавиатуры и мыши; правила оформления текстов помощи; перечень стандартных сообщений; правила обработки реакции пользователя?

A) ИСО 6385:1981, Эргономические принципы проектирования рабочих систем

B) ИСО 6592:1985, Обработка информации. Руководящие указания по документированию автоматизированных прикладных систем

C) ISO/IEC 2382-20:1990, Information technology ~ Vocabulary – Part 20 : Systems development. (Разработка систем)

2. Какой из перечисленных стандартов является стандартом оформления проектной документации?

A) ИСО 6385:1981, Эргономические принципы проектирования рабочих систем

B) ИСО 6592:1985, Обработка информации. Руководящие указания по документированию автоматизированных прикладных систем

C) ISO/IEC 2382-20: 1990, Information technology ~ Vocabulary - Part 20 : Systems development. (Разработка систем)

3. Как называют модель процесса, в которой число итераций возрастает настолько, что каждая новая итерация предоставляет слишком малое количество новых возможностей по сравнению с предыдущей?

A) каскадная (водопадная) модель

B) спиральная модель

C) инкрементальная модель

4. _____ - это непрерывный процесс, который начинается с момента принятия решения о необходимости создания программного обеспечения и заканчивается в момент его полного изъятия из эксплуатации.

5. Что понимают под термином программный продукт?

A) набор компьютерных программ, процедур и, возможно, связанных с ними документации и данных, предназначенных для передачи пользователю

B) набор компьютерных программ, процедур и, возможно, связанных с ними документации и данных, предназначенных для передачи пользователю

C) полный набор или часть программ, процедур, правил и связанной с ними документации системы обработки информации

6. При каком виде тестирования имеется доступ к программному обеспечению только через те же интерфейсы, что и для заказчика или пользователя, либо через внешние интерфейсы, позволяющие другому компьютеру либо другому процессу подключиться к системе для тестирования. _____

7. _____ - методология объектно-ориентированного проектирования, предназначенная для представления жизненного цикла объектов в реальном или абстрактном мире.

10. **УЧЕБНО-МЕТОДИЧЕСКОЕ И ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ (МОДУЛЯ)**

ОСНОВНАЯ ЛИТЕРАТУРА

1 Гамма Э. Приемы Объектно-ориентированного проектирования. Паттерны проектирования. / Э. Гамма. – Спб: Питер, 2008. – 366 с.

2 Лаптев, В.В. С++ Объектно-ориентированное программирование: учеб. пособие /

В.В. Лаптев – СПб.: Питер, 2008. – 458 с.

3 Пайлок, Д. Управление разработкой программного обеспечения. / Д. Пайлок. – СПб:Питер, 2011

ДОПОЛНИТЕЛЬНАЯ

1 Аллен, Э. Типичные ошибки проектирования. / Э. Аллен. – Спб: Питер, 2003. – 224 с.

2 Брауде, Э. Технология разработки программного обеспечения. / Э. Брауде – Спб: Питер, 2004. – 655 с.

3 Орлов, С.А. Технология разработки программного обеспечения. Учебн. Пособие. Рек. Мин. обр. РФ. / С.А. Орлов. – Спб: Питер, 2003, 2004. – 527 с.

ИНТЕРНЕТ-РЕСУРСЫ

	Наименование ресурса	Характеристика
1	http://www.intuit.ru	ИНТУИТ - сайт, который предоставляет возможность дистанционного обучения по нескольким образовательным программам, касающимся, в основном, информационных технологий. Содержит несколько сотен открытых образовательных курсов.
2	http://ru.wikipedia.org	Википендия – свободная общедоступная мультиязычная универсальная интернет-энциклопедия. Поиск по статьям, написанным на русском языке. Избранные статьи, ссылки на тематические порталы и родственные проекты.
3	http://www.biblioclub.ru	Электронная библиотечная система «Университетская система -online» специализируется на учебных материалах для ВУЗов по научно-гуманитарной тематике, а также содержит материалы по точным и естественным наукам.
4	http://www.window.edu.ru	Единое окно доступа к образовательным ресурсам/ каталог/ профессиональное образование

ПЕРИОДИЧЕСКИЕ ИЗДАНИЯ

Журналы «Программные продукты и системы», «Открытые системы», «Стандарты и качество».

11. МАТЕРИАЛЬНО-ТЕХНИЧЕСКОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ (МОДУЛЯ)

Мультимедийная лекционная аудитория (331)

В качестве программного обеспечения используются свободно распространяемое программное обеспечение Dev C++.

12. РЕЙТИНГОВАЯ ОЦЕНКА ЗНАНИЙ СТУДЕНТОВ ПО ДИСЦИПЛИНЕ

Балльная структура оценки за семестр

Семестровый модуль дисциплины					
Учебные модули	Виды контроля	Сроки выполнения	Макс. кол-во бал-	Посещение	Макс. кол-во баллов

			(недели)	лов	заня-тий, актив-ность	за уч. модуль
1	Этапы разработки программного обеспечения	Отчет по лаб. работе	1 – 4	8	2	10
2	Методы разработки программного обеспечения	Отчет по лаб. работе	5 – 10	7	3	10
3	Проектирование программного обеспечения на основе объектно-ориентированного подхода	Отчет по лаб. работе Тест №1	11 – 12	8 5	1	14
4	Документация по сопровождению программных средств	Отчет по лаб. работе	13 – 15	8	2	10
5	Международные стандарты разработки программного обеспечения	Тест №2	16 – 17	10 5	1	16
	Сдача зачета					40
	Итого					100

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ПРОВЕДЕНИЮ ПРАКТИЧЕСКИХ ЗАНЯТИЙ

Практические занятия являются одним из основных этапов в процессе изучения дисциплины.

Можно рекомендовать следующие основные этапы проведения занятия:

1 Организационный момент: взаимное приветствие преподавателя и студентов, проверка отсутствующих, организация внимания.

2 Постановка целей занятия: обучающей, развивающей, воспитывающей.

3 Планируемые результаты обучения: что должны студенты знать и уметь.

4 Проверка знаний: устный опрос, фронтальный опрос, письменный опрос, комментирование ответов, оценка знаний, обобщение по опросу.

5 Изучение нового материала по теме:

организация внимания;
проблемная ситуация;
объяснение, беседа;
связь с предыдущим материалом;
использование технических средств обучения;
анализ межпредметных связей;
развитие умственных способностей студентов в процессе объяснения, обобщения.

6 Закрепление материала предназначено для того, чтобы студенты запомнили материал и научились использовать полученные знания (активное мышление).

Формы закрепления:

решение задач;
групповая работа (коллективная мыслительная деятельность).

7 Домашнее задание:

работа над текстом учебника;
выполнение упражнений и решение задач;
подготовка текстов докладов и презентаций.

Тематика практических занятий:

1. Основные определения: программные средства, программное обеспечение, программный продукт. Классификация типов программного обеспечения. Понятие жизненного цикла программного обеспечения.

2. Модели жизненного цикла программного обеспечения: каскадная, спиральная, инкрементальная. Этапы разработки программного обеспечения.

3. Методы разработки программного обеспечения. Проект. Состав и структура коллектива разработчиков, их функции.

4. Нисходящий анализ процесса управления проектированием программного изделия. Восходящее проектирование.

5. Метод последовательной модернизации. Блок-схемы, ER-диаграммы, UML-диаграммы, DFD-диаграммы,

6. Проектирование программного обеспечения на основе объектно-ориентированного подхода. Конструирование программных систем как структурных коллекций, реализующих абстрактные типы данных.

7. Документация по сопровождению программных средств. Стандарты и практические руководства.

8. Организация выпуска документации в фазах исследования и анализа осуществимости. Организация выпуска документации в фазах конструирования и программирования. Организация выпуска документации в фазах оценки и использования.

9. Международные стандарты проектирования, разработки, оформления документации, пользовательского интерфейса.

На каждую из отведенных тематик отводится 2 часа аудиторного времени.

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ПРОВЕДЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

Лабораторные занятия - это групповые занятия со студентами под руководством преподавателя, но, в отличие от практических, на таких занятиях студенты преимущественно решают задачи с применением средств вычислительной техники.

Перед созданием любой программы требуется точно продумать алгоритм. Записать его блок-схемой или словесно. Надо четко определить, что в нее требуется ввести и что получить в результате, в какой последовательности выполнять действия. В случае необходимости выделить циклические структуры и подпрограммы. В циклах четко определить параметры, задать их начальные значения, определить условия повторения и завершения цикла. В функциях определить количество передаваемых и возвращаемых значений.

При кодировании программы нужно определить тип используемых данных в зависимости от возможного диапазона принимаемых значений. При вводе величины не забывать осведомить об этом пользователя, а иногда сообщить и о типе, диапазоне или порядке ввода значений. Такое сообщение должно быть информативно и коротко. Вывод данных лучше сопровождать текстом и форматированием. Формат вывода можно уточнить при помощи модификаторов.

В именах переменных необходимо отражать их назначение, что повышает читаемость и понимание программы.

При записи сложных выражений нужно обращать внимание на приоритет операций. Текст программы лучше сопровождать краткими и информативными комментариями, что облегчает как понимание программы, так и ее отладку.

Объявление локальных переменных предпочтительнее по сравнению с глобальными.

Для отладки программы нужно запустить ее на выполнение несколько раз, задавая различные значения вводимых величин. Перед запуском необходимо иметь заранее подготовленные тестовые примеры, содержащие исходные данные и ожидаемые результаты. Их количество зависит от алгоритма. Проверьте реакцию программы на заведомо неверные исходные данные.

Для быстрого поиска ошибки в алгоритме рекомендуется выводить промежуточные данные.

При сдаче лабораторной работы студент должен продемонстрировать преподавателю созданную программу, правильно работающую, отлаженную.

Преподаватель, принимая лабораторную работу, тестирует программу студента и задает ему вопросы по конструкциям, используемым в программе и теоретическим основам программирования.

МЕТОДИКА ВЫПОЛНЕНИЯ И ЗАДАНИЯ ЛАБОРАТОРНЫХ РАБОТ

1. Методы написания программного обеспечения

Этапы разработки программ
1 Постановка задачи
1.1 Формулировка и анализ физической задачи
1.2. Составление математической модели
1.3 Составление алгоритма задачи
2 Создание программы
2.1 Составление текста программы
2.2 Синтаксическая отладка программы
2.3 Тестирование и семантическая отладка
3 Документирование программы

3.1 Пользовательская документация программы
3.2 Документация по сопровождению программы
3.4 Запуск готовой программы и анализ полученных результатов

Создание Web-приложений. Работа с окнами средствами JavaScript

Главное окно браузера создается автоматически при запуске браузера. С помощью сценария можно создать любое количество окон, а также разбить окно на несколько прямоугольных областей, называемых фреймами. Окну браузера соответствует объект window, а HTML-документу, загруженному в окно, соответствует объект document. Эти объекты могут содержать в себе другие объекты. В частности, объект document входит в состав window.

Доступ к свойствам и методам данного объекта происходит, как и в других объектах, через точку. Поскольку объект document является подобъектом объекта window, ссылка на HTML-документ, загруженный в текущее окно: window.document. Объект document имеет метод write (запись строки в текущий HTML-документ).

Для его применения используют window.document.write(строка). Объект окна window - корневой объект, имеющий свои подобъекты. Например, location хранит информацию об URL-адресе загруженного документа, screen – данные о возможностях экрана монитора пользователя.

В объектной модели документа объекты сгруппированы в *коллекции*. Коллекция – промежуточный объект, содержащий объекты собственно документа. Коллекция является упорядоченным массивом объектов, отсортированных в порядке упоминания соответствующих им элементов в HTML-документе. Индексация объектов в коллекции начинается с нуля. Синтаксис обращения к элементам коллекции аналогичен синтаксису обращению к элементам массива. Коллекция имеет длину – свойство length.

Коллекция всех графических изображений документа называется images, коллекция всех форм – forms, ссылок – links. Коллекция всех объектов документа называется all.

Один и тот же объект может входить в частную коллекцию (например, images), но он обязательно входит в коллекцию all. При этом его индексы могут быть разными в разных коллекциях.

При использовании документа, загруженного в текущее окно, объект window можно не упоминать, а сразу начинать с объекта document.

Например,

document. images(0)

Вместо индекса можно использовать значение атрибута ID в теге, который определяет соответствующий элемент HTML-документа.

Однако универсальный способ обращения к объектам документа – обращение посредством коллекции all.

С помощью сценария можно создавать любое количество окон. Для этого применяется метод open( ):

window. open(параметры)

Данному методу передаются следующие необязательные параметры:

адрес документа, который нужно загрузить в создаваемое окно;

имя окна (как имя переменной);

строка описания свойств окна (features).

В строке свойств записываются пары свойство = значение, которые отделяются друг от друга запятыми.

Свойства, передаваемые в строке features

Свойство	Значения	Описание
channel mode	yes, no, 1, 0	Показывает элементы управления channel

directories	yes, no, 1, 0	Включают кнопки каталога
ful1screen	yes, no, 1, 0	Полностью разворачивает окно
height	число	Высота окна в пикселях
left	число	Положение по горизонтали относительно левого края экран в пикселях
location	yes, no, 1, 0	Текстовое поле Address
menubar	yes, no, 1, 0	Стандартное меню браузера
resizeable	yes, no, 1, 0	Возможность пользователя изменять размер окна
scrollbars	yes, no, 1, 0	Горизонтальные и вертикальные полосы прокрутки
		Стандартная строка состояния
status	yes, no, 1, 0	Включает панели инструментов браузера
toolbar	yes, no, 1, 0	Положение по вертикали относительно верхнего края экрана в пикселях
top	число	Ширина окна в пикселях
width	число	

Примеры

window. open (“mypage.htm”, “NewWin”, “height=150, width=300”)
window. open (“mypage.htm”)
strfeatures = “top=100, left=15, height=250, width=300, location=no”
window. open (“www.amsu.ru”, strfeatures)

Вместо строки strfeatures можно использовать значение true, тогда указанный документ загружается в существующее окно, вытесняя предыдущий документ.

Метод window.open() возвращает ссылку на объект окна, сохранив которую, можно использовать позднее, например, при закрытии окна.

Для закрытия используют метод close(). Однако, выражение window.close() закрывает главное окно. Для закрытия других окон используют ссылки.

Пример

var str = window. open (“mypage.htm”, “моя страница”)
str.close()

Объект document является центральным в иерархической объектной модели. Он предоставляет всю информацию о HTML-документе с помощью коллекций и свойств и множество методов для работы с документами.

Коллекция document

all	все теги и элементы основной части документа
anchor	якоря (закладки) документа
applets	все объекты документа, включая встроенные элементы управления, графические элементы, апплеты, внедренные объекты
embeds	все внедренные объекты документа
forms	все формы на странице
frames	фреймы, определенные в теге <FRAMESET>
images	графические элементы
links	ссылки и блоки <AREA>
plugins	другое название внедренных документов
scripts	все разделы <SCRIPT> на странице
styleSheets	контейнерные свойства стиля, определенные в документе

Методы document

clear	очищает выделенный участок
close	закрывает текущее окно браузера
createElement	создает экземпляр элемента для выделенного тега

elementFromPoint	возвращает элемент с заданными координатами
execCommand	выполняет команду над выделенной областью
open	открывает документ
queryCommandEnabled	сообщает, доступна ли данная команда
queryCommandIndeterm	сообщает, если данная команда имеет неопределенный статус
queryCommandState	возвращает текущее состояние команды
queryCommandSupported	сообщает, поддерживается ли данная команда
queryCommandText	возвращает строку, с которой работает команда
queryCommandValue	возвращает значение команды, определенное для документа или объекта TextRange
write (writeln)	записывает текст и код HTML в документ, находящийся в указанном окне

Свойства document

Свойство	Атрибут	Назначение
activeElement		Активизирует активный элемент
alinkColor	ALINK	Цвет ссылок на странице
bgColor	BGCOLOR	Определяет цвет фона элемента
body		Ссылка только для чтения на неявный основной объект документа, определенный в теге <BODY>
cookie		Строка cokie-записи. Значение этого свойства приводит к записи на диск.
domain		Устанавливает или возвращает домен документа для его защиты или идентификации.
fgColor	TEXT	Устанавливает цвет текста переднего плана
lastModified		Дата последней модификации страницы, доступна как строка
linkColor	LINK	Цвет еще непосещенных гиперссылок на странице
location		Полный URL документа
parentWindow		Возвращает родительское окно для документа
readyState		Определяет текущее состояние загружаемого объекта
referer		URL страницы, которая вызвала текущую
selection		Ссылка только для чтения на дочерний для document объект selection
title	TITLE	Определяет справочную информацию элемента, используемую при загрузке и всплывающей подсказке
url	URL	URL-адрес документа клиента или в теге <META>
vlinkColor	VLINK	Цвет посещенных ссылок на странице

Объект window кроме дочерних объектов имеет свои методы, свойства, события.

Свойства window:

parent	возвращает родительское окно для текущего
self	возвращает ссылку на текущее окно
top	возвращает ссылку на главное окно
name	название окна
opener	окно, создаваемое текущим
closed	сообщает, если окно закрыто
status	текст, показываемый в строке состояния браузера

defaultStatus	текст по умолчанию строки состояния браузера
returnValue	позволяет определить возвращаемое значение для события или диалогового окна
client	ссылка, возвращаемая объект навигатора браузеру
document	ссылка только для чтения на объект окна document
event	ссылка только для чтения на глобальный объект event
history	ссылка только для чтения на объект окна history
location	ссылка только для чтения на объект окна location
navigator	ссылка только для чтения на объект окна navigator
screen	ссылка только для чтения на объект окна screen

Например,

window.status = «работает сценарий»

Свойство parent позволяет обратиться к объекту, расположенному в иерархии на одну ступень выше. Для перемещения на две ступени выше используют parent.parent. Для обращения к самому главному окну – окну браузера, используют свойство top.

Свойство status используют для вывода сообщений во время работы сценария. Например, window. status = “сценарий работает”

Методы window

open( )	открывает новое окно браузера
close( )	закрывает текущее окно браузера
showHelp( )	показывает окно подсказки как диалоговое
showModalDialog( )	показывает новое модальное(диалоговое) окно
alert( )	окно предупреждения с сообщением и кнопкой ОК
prompt( )	окно приглашения с сообщение, текстовым полем и кнопками ОК и Cancel (Отмена)
confirm( )	окно подтверждения с сообщением и кнопками ОК и Cancel
navigate( )	загружает другую страницу с указанным адресом
blur( )	убирает фокус с текущей страницы
focus( )	устанавливает страницу в фокус
scroll( )	разворачивает окно на заданную ширину и высоту
setInterval( )	указывает процедуре выполняться автоматически через заданное число миллисекунд
setTimeout( )	запускает программу через заданное количество миллисекунд после загрузки страницы
clearInterval( )	обнуляет таймер, заданный методом setInterval( )
clearTimeout( )	обнуляет таймер, заданный методом setTimeout( )
execScript( )	выполняет код сценария, по умолчанию Jscript

Рассмотренные выше методы позволяют работать с независимыми (немодальными) окнами. Для создания модального окна используется метод showModalDialog(). В качестве параметра данный метод принимает адрес документа (файла), имя окна, и строку свойств.

При работе с модальными окнами пользователь не может обратиться к другим окнам, в том числе и к главному. Окна, создаваемые методами alert(), promt(), confirm() являются модальными.

Одним из главных назначений сценариев в HTML-документе является обработка событий, таких как щелчок кнопки мыши по элементу документа, помещение указателя мыши на элемент, нажатие клавиши и др. Для одного и того же элемента можно определить несколько событий на которые он будет реагировать.

Сообщение о событии формируется в виде объекта, т.е. контейнера для хранения информации. Объект события в одном из свойств содержит ссылку на элемент, с которым связано данное событие (на кнопку, изображение и т.п.)

Обычно обработчики событий оформляются в виде функций, определения которых помещаются в контейнерный тег <SCRIPT>.

События window

onblur	выход окна из фокуса
onfocus	окно становится активным
onhelp	нажатие пользователем клавиши F1
onresize	изменение пользователем размеров окна
onscroll	прокрутка окна пользователем
onerror	ошибка при передаче
onbeforeunload	для сохранения данных перед выгрузкой страницы
onload	страница полностью загружена
onunload	непосредственно перед выгрузкой страницы

В случае открытия нескольких окон браузера, пользователь может переключаться между ними, переводя фокус с одного окна на другое. Эти действия инициируются программными событиями onblur и onfocus. Эти же действия можно вызвать, используя методы blur и focus.

Событие onerror происходит при ошибки загрузки страницы или ее элемента. Его можно использовать в программе при попытке вновь загрузить страницу. Например,

```
<SCRIPT>
function window.onerror( ) {
alert (“ Ошибка! Повтори попытку!”)
}
</SCRIPT>
```

События document

onafterupdate	окончание передачи данных
onbeforeupdate	перед выгрузкой страницы
onclick	при щелчке левой кнопкой мыши
ondblclick	при двойном щелчке левой кнопкой мыши
ondragstart	при возникновении перетаскивания
onerror	ошибка при передаче
onhelp	нажатие клавиши F1
onkeydown	нажатие клавиши
onkeypress	возникает при нажатии клавиши и продолжается при удержании клавиши в нажатом состоянии
onkeyup	пользователь отпускает клавишу
onload	при полной загрузке документа
onmousedown	при нажатии кнопки мыши
omousemove	при перемещении указателя мыши
onmouseout	когда указатель мыши выходит за границы элемента
onmouseover	когда указатель мыши входит на документ
onmouseup	пользователь отпускает кнопку мыши
onreadystatechange	возникает при изменении свойства readystate
onselectstart	когда пользователем впервые запускается выделенная часть документа

Создание Web-приложений. Динамическое изменение элементов документа

Элементы HTML-документа задаются тегами, большинство из которых имеют параметры (атрибуты). В объектной модели документа тегам соответствуют объекты, а ат-

рибутам – свойства этих объектов. Названия свойств объектов, как правило, совпадают с названиями атрибутов, но записываются в нижнем регистре.

Наиболее удобный способ динамического изменения HTML-документа основан на использовании свойств innerText, outerText, innerHTML и outerHTML. С их помощью можно получить доступ к содержимому элемента. Изменяя значения перечисленных свойств можно частично или полностью изменить сам элемент. Например, можно изменить только надпись на кнопке, а можно превратить кнопку в изображение или Flash-анимацию.

Значением свойства innerText является все текстовое содержимое между открывающим и закрывающим тегами элемента. Внутренние теги игнорируются. Данные открывающего и закрывающего тегов соответствующего элемента также не входят.

В отличие от предыдущего свойство outerText включает в себя данные открывающего и закрывающего тегов. Таким образом, outerTextесть весь текст, содержащийся в контейнере, включая его внешние теги. Например, задан HTML-код:

<DIV ID = “my” >
<A HREF = ‘raznoe.htm’>
<IMG SRC = ‘picture.jpg> Ссылка на раздел <B> Разное <\B>
</A>
<\DIV>

Здесь свойства innerText и outerText для элемента, заданного контейнерным тегом <DIV>, совпадают:

document.all.my.innerText //значение равно – «Ссылка на раздел Разное»

При присвоении свойствам innerText и outerText новых значений нужно помнить, что если значения содержат теги, то они не интерпретируются, а воспринимаются как обычный текст.

Свойство innerHTML содержит внутренний HTML-код контейнера элемента. Присвоение этому свойству нового значения, содержащего HTML-код, приводит к интерпретации кода. Свойство outerText дополнительно включает внешние открывающие и закрывающие теги элемента.

Для приведенного HTML-кода значение document. all. my. innerHTML равно “<A HREF = ‘raznoe.htm’> <IMG SRC = ‘picture.jpg’> Ссылка на раздел <B> Разное <\B> </A>”/

Значение document. all. my. outerHTML – “<DIV ID = “my” > <A HREF = ‘raznoe.htm’> <IMG SRC = ‘picture.jpg> Ссылка на раздел <B> Разное <\B>
</A> <\DIV>”.

Если в сценарии выполнить выражение

document.all.my.innerHTML = “<BUTTON>Щелкни здесь</BUTTON>”

ссылка, изображение и текст будут заменены кнопкой с надписью «Щелкни здесь». При этом контейнерный тег “<DIV ID = “my” > сохранится. Если аналогичным образом использовать outerHTML, кнопка также появится, но уже без контейнера “<DIV ID = “my” >.

Свойства innerHTML и outerHTML могут применяться к элементам, заданным неконтейнерными тегами. Тогда innerHTML и outerHTML совпадают.

Для ускорения загрузки графики можно использовать следующие возможности JavaScript. Можно организовать предварительную загрузку изображений в кэш-память браузера, не отображая их на экране. Это особенно эффективно при начальной загрузке страницы. Пока изображения загружаются в память, оставаясь невидимыми, пользователь может рассматривать текстовую информацию.

Для предварительной загрузки изображения требуется создать его объект в памяти браузера. Это можно сделать следующим выражением:

myimg = new Image (ширина, высота)

Параметры должны соответствовать значениям атрибутов WIDTH и HEIGHT тега <IMG>, который используется для отображения предварительно загруженного изображения.

Для созданного в памяти объекта изображения можно создать имя или URL-адрес графического файла:

myimg.scr = “URL-адрес изображения”

что предписывает браузеру загрузить изображения без его отображения.

После загрузки в кэш-память всех изображений и загрузки всего документа можно сделать их видимыми. Для этого свойству src элемента <IMG> нужно присвоить значение этого же свойства объекта изображения в кэш-памяти. Например,

document. images[0]. scr = myimg. scr

Здесь слева указано свойство scr первого в документе элемента, соответствующего тега <IMG>, справа – свойство scr объекта изображения в кэш-памяти.

С помощью JavaScript можно через заданный интервал времени запускать код или функцию. При этом создается эффект одновременного (параллельного) выполнения вычислительных процессов.

Для организации повторения через заданный интервал выполнения некоторого выражения служит метод setInterval( ) объекта window:

setInterval(выражение, период, [, язык])

Первым параметром является строка, например вызов функции. Период указывается в миллисекундах. Третий параметр – необязательный, в котором указывается язык с помощью которого написано заданное выражение. По умолчанию – JavaScript.

Метод setInterval( ) возвращает некоторое целое число – идентификатор временного интервала, который может быть использован в дальнейшем, например для прекращения выполнения процесса методом clearInterval( ).Например,

```
var pr = setInterval( “myfunc(), 100” )
if (confirm ( “Прервать процесс?”) )
clearInterval(pr)
```

Если требуется выполнить действие с некоторой временной задержкой, используется метод setTimeout( ), имеющий синтаксис аналогичный setInterval( ). Для отмены задержки процесса, запущенного setTimeout( ), используют clearTimeout( ).

Задание

Создать HTML-документ, расположив в нем список названий графических объектов, одно исходное отображение, две кнопки.

Щелчок на элементе списка должен приводить к изменению цвета элемента списка и отображению соответствующего графического элемента, и соответствующего ему тестового сопровождения.

При этом изображение кнопки должно быть также изменено.

Щелчок по первой кнопке через 5 секунд должен инициализировать функцию открытия документа в окне, заданного размера, определенного размера текстового поля и название. Окно должно содержать горизонтальные и вертикальные полосы прокрутки, размер окна не должен изменяться по желанию пользователя. Выведенный в окне текст должен быть синим на сером фоне, иметь выделенный заголовок, ссылки на другие объекты. По выбору продемонстрируйте по пять событий и свойств объектов window и document.

Это действие может быть отменено с помощью второй кнопки.

Создание Web-приложений. Работа с фреймами

Фрейм – прямоугольная область окна браузера, в которую можно загрузить HTML-документ. Разбиение окна браузера на отдельные окна производится с помощью тега <FRAMESET>, внутрь которого вставляются теги <FRAME> с атрибутами, указывающи-

ми имя фрейма и адрес HTML-документа.

Пример

```
<HTML>
<FRAMESET ROWS= “30%, 70%”>
<FRAMESET SRC= “документ1.htm” NAME = “frame1” >
<FRAMESET SRC= “документ2.htm” NAME = “frame2” >
</FRAMESET>
</HTML>
```

Здесь применяется вертикальное расположение фреймов. Для горизонтального размещения фреймов вместо атрибута ROWS в теге <FRAMESET> использовать COLS.

Используя вложение тега <FRAMESET>, можно разбить уже имеющийся фрейм на два других.

При разбиении окна на фреймы и, в свою очередь, фрейма на другие фреймы возникают отношения родитель-потомок. Каждому из фреймов соответствует свой объект document. Обеспечение доступа к в иерархии объектов представлено на рисунке.

Так при обращении из одного фрейма-потомка к другому, необходимо помнить, что прямой связи между фреймами-потомками не существует. Поэтому сначала нужно обратиться к родительскому окну, а затем к его второму потомку:

parent. frame2. document. write(“ Привет от первого фрейма.“)

Можно изменить элемент одного фрейма из другого. Например, при щелчке на тексте в правом фрейме в левом изменится один из текстовых документов. Тогда документ в левом фрейме с именем LEFT:

```
<HTML>
Делай раз<BR>
Делай два
<H1 ID = "XXX"> Делай три </H1>
</HTML>
```

Документ в правом фрейме:

```
<HTML>
<SCRIPT>
function change( ) {
parent.LEFT.document.all.XXX. innerText = "Делай пять!!!!"
}
</SCRIPT>
<H1 onclick = "change( )"> Щелкни здесь</H1>
</HTML>
```

В теле функции change( ) происходит обращение к левому фрейму с именем LEFT (задается в установочном HTML-файле) через parent. Изменение элемента происходит за счет присвоения значения свойству innerText. Кроме данного свойства можно использовать outerText, innerHTML или outerHTML.

Важно, что изменения в одном фрейме по событию в другом происходят без перезагрузки HTML-документа.

Фреймы удобно использовать при создании навигационных панелей. В одном фрейме располагаются ссылки, а второй предназначен для отображения документов, вызываемых при активизации соответствующих ссылок.

Пример

```
// установочный файл frame.htm
<HTML>
<FRAMESET COLS = “25%, 75%”>
<FRAME SRC = “menu.htm” NAME = “menu” >
<FRAME SRC = “start.htm” NAME = “main” >
</FRAMESET>
```

```
</HTML>
```

Здесь start.htm – документ, который первоначально показан во фрейме main.

```
//menu.htm – навигационная панель
<HTML>
<SCRIPT>
function load (url) {
parent. main. location. href = url;
}
</SCRIPT>
<BODY>
<A HREF = “javascript:load(‘первый.htm’)”>Первый </A>
<A HREF = “второй.htm” TARGET = “main”> Второй </A>
<A HREF = “третий.htm” TARGET = “top”> Третий </A>
</BODY>
</HTML>
```

В примере окно браузера разделено на два фрейма. Первый из них играет роль навигационной панели, а второй – окна для отображения документов. Продемонстрированы два способа загрузки новой страницы во фрейм main. В первом случае используется функция load( ), параметр которой указывает, какой файл следует загрузить. При этом место, в которое он загружается, определяется самой функцией load( ). Во второй ссылке используется атрибут TARGET. В третьем ссылке демонстрируется, как можно избавиться от фреймов.

Для удаления фрейма с помощью load( ) достаточно записать:

parent. location. href = url

Атрибут TARGET в теге ссылки <A HREF> обычно применяется в случаях, когда требуется загрузить одну страницу в один фрейм. Язык сценариев используют при необходимости выполнения нескольких действий.

Для ссылок из родительского окна к объектам его дочерних фреймов можно использовать коллекцию frames. Обращение к определенному фрейму из этой коллекции возможно по индексу или по имени фрейма:

window. frames [индекс]

window. имя_фрейма

При обращении к объекту документа, загруженного во фрейм, следует сначала упомянуть объект document:

window. frames(0). document. all. Myinput. Value

window. LEFT. document. all. Myinput. Value

Ссылка из дочернего фрейма на родительский - осуществляется с использованием parent.

При использовании top следует учитывать, что создаваемый сайт может быть загружен в другой. Тогда объект top окажется объектом другого сайта. Поэтому лучше использовать parent для ссылок на вышестоящее окно или фрейм.

Ссылки top или self используют для предотвращения отображения сайта внутри фреймов другого сайта. Сценарий, выполняющий это, следует разместить в начале документа, например:

```
<SCRIPT>
if (top != self )
        top. Location = location
</SCRIPT>
```

т.е., ссылка на свойство top на верхнее окно, должна совпадать со ссылкой self на текущее окно.

Для вставки одного HTML-документа в тело другого средствами браузера служит контейнерный тег <IFRAME>:

<IFRAME SRC = “адрес документа” > </IFRAME>

Данный элемент представляет собой прямоугольную область с прокруткой или без. Такое окно называют плавающим фреймом. Данный документ можно позиционировать с помощью параметров таблицы стилей (тег <STYLE> или атрибут STYLE).

Плавающий фрейм аналогичен обычному фрейму. При создании он помещается в коллекцию frames. Среди его свойств широко используется align – выравнивание плавающего фрейма относительно окружающего содержимого документа. Его возможные значения:

absbotton – выравнивает нижнюю границу фрейма по подстрочной линии символов окружающего текста,

absmiddle – выравнивает середину границу фрейма по центральной линии между top и absbotton окружающего текста,

baseline – выравнивает нижнюю границу фрейма по базовой линии окружающего текста,

botton – совпадает с baseline (только IE)

left – выравнивает фрейм по левому краю элемента-контейнера,

middle – выравнивает воображаемую центральную линию окружающего текста по воображаемой центральной линии фрейма,

right – выравнивает фрейм по правому краю элемента-контейнера,

texttop – выравнивает верхнюю границу фрейма по надстрочной линии символов окружающего текста,

top – выравнивает верхнюю границу фрейма по верхней границе окружающего текста.

Задание

Вариант 1. Окно браузера поделить на два фрейма. В левом расположить небольшое изображение. Организовать возможность вывода полномасштабного изображения в левом окне по щелчку мыши на миниатюре изображения в правом фрейме. В левом фрейме дополнительно организовать навигационную модель. Создать новый HTML-документ и вставить его в ранее созданный, как плавающий вертикальный фрейм, выравнивая нижнюю границу фрейма по базовой линии окружающего текста.

Вариант 2. Окно браузера поделить на три фрейма. В левом расположить небольшое изображение. Организовать возможность вывода полномасштабного изображения в среднем окне по щелчку мыши на миниатюре изображения в правом фрейме. В левом фрейме дополнительно организовать навигационную модель. Создать новый HTML-документ и вставить его в ранее созданный, как плавающий вертикальный фрейм, выравнивая воображаемую центральную линию окружающего текста по воображаемой центральной линии фрейма.

Создание Web-приложений. Простые визуальные эффекты

Смена изображений

Для смены одного изображения на другое достаточно с помощью сценария заменить значение атрибута SCR тега <IMG>. Например:

```
<HTML>
<IMG ID = “myimg” SRC= ‘pict1.gif ’
onclick = “document.all.myimg.src = ‘pict2.gif’ ”>
</HTML>
```

Здесь смена изображения из файла pict1.gif на изображение из файла pict2 происходит при первом щелчке на нем. Последующие щелчки не приведут к видимым изменениям, поскольку второе изображение будет заменяться им же. Чтобы при повторном щелчке происходила замена изображения на предыдущее необходимо создать перемен-

ную-триггер (флаг), принимающий одно из двух возможных значений, по которому можно определить, какое из двух значений надо отобразить.

```
<HTML>
<IMG ID = “myimg” SRC= ‘pict1.gif ’
onclick = “ imgchange( )”>
<SCRIPT>
var flag=false
function  imgchange( ) {
if (flag) document. all. myimg. src = “pict1.gif ”
        else document. all. myimg. src = “pict2.gif ”
flag=!flag
}
</SCRIPT>
</HTML>
```

Цветовые эффекты

Задача изменения цвета кнопки при наведении на нее указателя мыши и возвращения в первоначальное состояние при удалении указателя с кнопки может быть решена следующим образом:

```
<HTML>
<STYLE>
mystile {font-weight.bold; background-color: a0a0ao} // серый цвет кнопок
</STYLE>

<FORM onmouseover = “colorchange ( ‘yellow’)” onmouseout
= “colorchange (‘a0a0ao’)”>
<INPUT TYPE = “BUTTON” VALUE = “Кнопка” CLASS = “mystile”
onclick = “alert( ‘Вы нажали кнопку’)” >
</FORM>
<SCRIPT>
function colorchange (color){
if (event. scrElement.type = = “button”)
        event. scrElement. style. backgroundColor = color;
}
</SCRIPT>
</HTML>
```

Функция colorchange( ) проверяет, является ли инициатор события объектом типа button. Если это так, то цвет кнопки меняется. Без этой проверки менялся бы не только цвет кнопок, но и текста.

Аналогичным способом можно изменять цвет фрагментов текста. Но в этом случае текст должен быть заключен в контейнер, например в теги <P>, <B>, <I>, <DIV>.

Можно создать прямоугольную рамку, окаймляющую текст, которая периодически изменяет цвет. Рамка создается тегами одноячеечной таблицы с заданием нужных атрибутов и параметров стиля:

```
<TABLE ID= “tab” BORDER=1 WIDH=200 style= “border:10 solid : red”>
<TR><TD> Доброе утро! </TR></TD>
</TABLE>
```

Функция изменения цвета:

```
<SCRIPT>
function flash( ) {
if ( !document.all) return null;
if (tab.style.borderColor = = ‘red’)   tab.style.borderColor = ‘yellow’
```

```
else tab.style.borderColor = ‘red’;
}
setInterval (“flash”, 500);                    //мигание рамки с интервалом 500 мс
</SCRIPT>
```

Объемные заголовки

Объемные заголовки часто используются на веб-страницах. Идея создания объемного заголовка состоит в наложении нескольких надписей с одинаковым содержанием с некоторым сдвигом по координатам. Наилучший эффект достигается путем подбора цветов надписей (игрой света и тени) с учетом цвета фона. Для этого используют библиотеку стилей. Функция, создающая заголовок с заданными параметрами:

```
        function d3 (text, x, y, tcolor, fsize, fweight, family, zind) {
/*      text – текс заголовка
        x – горизонтальная координата (left)
        y – вертикальная координата (top)
        tcolor – цвет переднего плана
        fsize – размер шрифта (пт)
        fweight – вес (толщина шрифта)
        family – название семейства шрифтов
        zind z-Index   */
if (!text) return null // если текст не указан ничего не выполняется
//значение параметров по умолчанию
if (!x) x=0
if (!y) y=0
if (!tcolor) tcolor=’00aaff ’
if (!fsize)  fsize=36
if (!fweight)  fweight =800
if (!family) ffamily=’arial’
// внутренние настройки
var sd=5, hd=2
var xzind= “ ”
if (zind) xzind= “; - Index:”+zind
var xstyle =’font-family:’ + family + ‘;font-size:’ + fsize + ‘;font-weight:’ + fweight + ‘;’
var xstr = ‘<DIV STYLE = “position: absolute; top:’  + (y +sd ) + ‘; left :’ +
( x + sd ) +  xzind + ‘ “>’
xstr+=‘<P styl e  = “ ’ + xstyle + ‘color: darked”>’ + text + ‘</P></DIV>’
xstr+= ‘<DIV STYLE = “ position: absolute; top:’ + y  + ‘; left :’ +
x  +  xzind + ‘ “>’
xstr+=‘<P styl e  = “ ’ + xstyle + ‘color: silver”>’ + text + ‘</P></DIV>’
xstr+= ‘<DIV STYLE = “ position: absolute; top:’ + ( y + hd )  + ‘; left :’ +
(x +hd)  +  xzind + ‘ “>’
xstr+=‘<P styl e  = “ ’ + xstyle + ‘color:’ + tcolor +   ‘”>’ + text + ‘</P></DIV>’
document.write(xstr)            //запись в документ
}
```

Параметр z-Index позволяет установить слой, в котором находится заголовок, и тем самым указать, будет ли заголовок располагаться над или под другим видимым элементом документа. Элементы с более высоким значением z-Index находятся над элементами, у которых z-Index меньше. Перекрытие элементов с одинаковыми значениями z-Index определяется порядком их следования в HTML-документе.

Вызов приведенной выше функции может выглядеть так:

```
d3 (“это не графика, это просто стиль текста”, 50, 50, ‘blue’, 72, 800, ‘times’)
```

Задание

Выполнить следующие действия на веб-странице:

1. Создать программу для работы с галереей миниатюр. При щелчке кнопкой мыши по миниатюре изображение должно увеличиваться, а затем при щелчке на увеличенном изображении оно должно уменьшаться. Доработайте приведенную в тексте функцию функцию imgchange( ). Для решения этой задачи потребуется массив флагов и функция обработчик, определяющая на каком именно изображении произошел щелчок:

```
var p1=new Array (“pict1.gif” , … )  //массив имен исходных файлов
var p2=new Array (“pict2.gif” , … )  //массив имен замещающих файлов
//формирование тегов, описывающих изображения
var xstr =  “ “
for (i=0; i<p1.length; i++)
xstr+= ‘<IMG ID = “i’ + i +’ ” SRC = “ ’+ p1[i]+’ ” onclick = “imgchange( )” >’
}
document.write(xstr)            // запись в документ
```

2.Выполнить замену фрагмента текста с черного на красный при наведении на него указателя мыши.

3. Выделите фрагмент текста мигающей трехцветной рамкой.

4.Создать эффект динамического изменения цвета ссылок. Различать цвета мерцания использованных и неиспользованных ссылок. (Множество цветов задать массивом. Использовать свойства linkColor и linkColor объекта document). Для изменения цвета случайным образом можно использовать метод random( ) (счетчик случайных чисел) встроенного объекта Math. Если требуется получить случайное число x, лежащее в интервале от A до B, то x=A+(B – A)*Math. random( )

5. Создать три объемных заголовка, поэкспериментировав со значениями внутренних параметров sd, hd, а также с заданием параметров по умолчанию.

Создание Web-приложений. Применение фильтров

С помощью фильтров каскадных таблиц стилей можно получить разнообразные эффекты: постепенное появление или исчезновение рисунка, плавное преобразование одного изображения в другое, задание степени прозрачности и др.

Фильтр следует понимать как некий инструмент преобразования изображения, взятого из графического файла и вставленного в HTML – документ с помощью тега <IMG>. Следует иметь в виду, что фильтры работают только в IE4+.

Фильтры можно применять не только к графическим объектам, но и к текстам, текстовым областям, кнопкам.

Прозрачность

С помощью фильтра alpha можно установить прозрачность графического объекта. Сквозь прозрачные графические объекты видны нижележащие изображения. Прозрачность имеет несколько вариантов градиентной формы. Например, она может увеличиваться от центра к краям изображения

Фильтр alpha задается с помощью каскадной таблицы стилей и имеет ряд параметров. В примере для графического изображения стиль определяется с помощью атрибута STYLE:

```
<IMG ID = “myimg” SCR = “ pict. gif ”
STYLE = “position: absolute; top:10; left: 50;
filter: alpha (opacity = 70, style = 3)”>
```

Здесь целочисленный параметр opacity определяет степень непрозрачности. Значение 0 соответствует полной прозрачности изображения, а 100 – полной непрозрачности. Параметр style задает градиентную форму распределения прозрачности по изображению

как целое число от 0 до 3. По умолчанию значение параметра равно 0, и градиент не применяется. Фильтр имеет и другие параметры, определяющие прямоугольную область изображения, к которому применяется фильтр. По умолчанию фильтр применяется ко всему изображению.

Фильтр можно определить в каскадной таблице силей внутри контейнерного тега <STYLE>:

```
<HTML>
<STYLE>
#myimg{position: absolute; top:10; left: 50; filter: alpha (opacity = 70, style = 3)}
</STYLE>
< IMG ID = “myimg” SCR = “ pict. gif ”
</HTML>
```

Доступ к свойствам фильтра в сценарии:

document. all. id_изображения. filters [“имя_фильтра”]. параметр = значение

Для рассмотренного примера это выражение имеет вид:

document. all. myimg. filters [“alpha”]. opacity = 30

Для остальных параметров alpha аналогично.

Для IE5.5+ можно использовать другой синтаксис, в котором в каскадной таблице стилей задается ссылка на специальный компонент и имя фильтра:

```
#myimg { filter: progid: DXImageTransform . Microsoft . alpha
(opacity = 70, style = 3)}
```

Тогда доступ к свойствам фильтра:

document.all.myimg.filters [“DXImageTransform. Microsoft alpha”]. opacity = 30

Трансформация

Фильтр alpha статический. Существуют и динамические фильтры: apply( ) – фиксирует изображение, play( ) – трансформирует, revealtrans( ) –преобразовывает изображение, stop( ) останавливает процесс преобразования при необходимости.

Фильтр revealtrans( ) имеет параметры: duration – длительность преобразования в секундах (число с плавающей точкой) и transition – тип преобразования (целое от 0 до 23).

Для эффекта появления изображения можно воспользоваться фрагментом, который происходит после загрузки документа, т.е. по событию onload:

```
<HTML>
<BODY onload = “transform()” >
<IMG ID = “myimg” SRC = “ pict. gif ” STYLE = “position: absolute; top:10;
left: 50; visibility = “hidden” filter: revealtrans (duration = 3, transition =12)” >
//transition =12 соответствует плавной трансформации
</BODY>
<SCRIPT>
function transform ( ) {                                    //появление изображения
document.all.myimg.style. visibility = “hidden”     // изображение невидимо
myimg. filters ( “revealtrans”). apply( )
myimg.style. visibility = “visible”
myimg. Filters ( “revealtrans”). play( )                    // выполняем преобразования
}
</SCRIPT>
</HTML>
```

Для замены одного изображения на другое необходимо установить начальное и конечное изображение путем присвоения нужных значений свойству src объекта, соответствующего изображению, например фрагментом:

document.all.myimg.src = “pict2. gif ”

Рассмотренный синтаксис воспринимается браузерами IE4+. Для IE5.5+ в каскад-

ной таблице стилей задается ссылка на специальный компонент и имя фильтра. Так для трансформации изображения по щелчку мыши на графическом объекте в другое, и обратно, можно воспользоваться программой:

```
<HTML>
<STYLE>
#myimg{position: absolute; top:10; left: 50; filter: progid: DXImageTransform . Microsoft revealtrans (duration = 3, transition = 12)}
</STYLE>
< IMG ID = “myimg” onclick = “transform( )” SCR = “ ear. gif ”>
<SCRIPT>
 function transform( ) {
//фиксация исходного изображения
myimg. filters (“DXImageTransform . Microsoft revealtrans”). apply ( )
//определение конечного изображения
if (document. all. mying. src. index0f (“ear”)!= -1)
document. all. mying. src = “s.gif”
        else document. all. mying. src = “ear.gif”
//выполняем преобразование
myimg. filters (“DXImageTransform . Microsoft revealtrans”). play ( )
}
</SCRIPT>
</HTML>
```

В браузере IE5.5+ возможно прменение фильтра basicimage, с помощью которого изображение можно повернуть на угол, кратный 90 градусам, задать прозрачность, зеркально отразить и др.

Задание

В HTML-документе из предыдущей лабораторной работы применить к рисункам методы трансформации и изменения прозрачности изображения и фона.

Создание Web-приложений. Динамическое изменение элементов документа

Элементы HTML-документа задаются тегами, большинство из которых имеют параметры (атрибуты). В объектной модели документа тегам соответствуют объекты, а атрибутам – свойства этих объектов. Названия свойств объектов, как правило, совпадают с названиями атрибутов, но записываются в нижнем регистре.

Наиболее удобный способ динамического изменения HTML-документа основан на использовании свойств innerText, outerText, innerHTML и outerHTML. С их помощью можно получить доступ к содержимому элемента. Изменяя значения перечисленных свойств можно частично или полностью изменить сам элемент. Например, можно изменить только надпись на кнопке, а можно превратить кнопку в изображение или Flash-анимацию.

Значением свойства innerText является все текстовое содержимое между открывающим и закрывающим тегами элемента. Внутренние теги игнорируются. Данные открывающего и закрывающего тегов соответствующего элемента также не входят.

В отличие от предыдущего свойство outerText включает в себя данные открывающего и закрывающего тегов. Таким образом, outerTextесть весь текст, содержащийся в контейнере, включая его внешние теги. Например, задан HTML-код:

```
<DIV ID = “my” >
<A HREF = ‘raznoe.htm’>
<IMG SRC = ‘picture.jpg> Ссылка на раздел <B> Разное <\B>
</A>
<\DIV>
```

Здесь свойства innerText и outerText для элемента, заданного контейнерным тегом <DIV>, совпадают:

document.all.my.innerText //значение равно – «Ссылка на раздел Разное»

При присвоении свойствам innerText и outerText новых значений нужно помнить, что если значения содержат теги, то они не интерпретируются, а воспринимаются как обычный текст.

Свойство innerHTML содержит внутренний HTML-код контейнера элемента. Присвоение этому свойству нового значения, содержащего HTML-код, приводит к интерпретации кода. Свойство outerText дополнительно включает внешние открывающие и закрывающие теги элемента.

Для приведенного HTML-кода значение document. all. my. innerHTML равно "<A HREF = 'raznoe.htm'> <IMG SRC = 'picture.jpg> Ссылка на раздел <B> Разное <\B> </A>"/

Значение document. all. my. outerHTML – "<DIV ID = "my" > <A HREF = 'raznoe.htm'> <IMG SRC = 'picture.jpg> Ссылка на раздел <B> Разное <\B>
</A> <\DIV>".

Если в сценарии выполнить выражение

document.all.my.innerHTML = "<BUTTON>Щелкни здесь</BUTTON>"

ссылка, изображение и текст будут заменены кнопкой с надписью Щелкни здесь. При этом контейнерный тег "<DIV ID = "my" > сохранится. Если аналогичным образом использовать outerHTML, кнопка также появится, но уже без контейнера "<DIV ID = "my" >.

Свойства innerHTML и outerHTML могут применяться к элементам, заданным неконтейнерными тегами. Тогда innerHTML и outerHTML совпадают.

Для ускорения загрузки графики можно использовать следующие возможности JavaScript. Можно организовать предварительную загрузку изображений в кэш-память браузера, не отображая их на экране. Это особенно эффективно при начальной загрузке страницы. Пока изображения загружаются в память, оставаясь невидимыми, пользователь может рассматривать текстовую информацию.

Для предварительной загрузки изображения требуется создать его объект в памяти браузера. Это можно сделать следующим выражением:

myimg = new Image (ширина, высота)

Параметры должны соответствовать значениям атрибутов WIDTH и HEIGHT тега <IMG>, который используется для отображения предварительно загруженного изображения.

Для созданного в памяти объекта изображения можно создать имя или URL-адрес графического файла:

myimg.scr = "URL-адрес изображения"

что предписывает браузеру загрузить изображения без его отображения.

После загрузки в кэш-память всех изображений и загрузки всего документа можно сделать их видимыми. Для этого свойству src элемента <IMG> нужно присвоить значение этого же свойства объекта изображения в кэш-памяти. Например,

document. images[0]. scr = myimg. scr

Здесь слева указано свойство scr первого в документе элемента, соответствующего тега <IMG>, справа – свойство scr объекта изображения в кэш-памяти.

С помощью JavaScript можно через заданный интервал времени запускать код или функцию. При этом создается эффект одновременного (параллельного) выполнения вычислительных процессов.

Для организации повторения через заданный интервал выполнения некоторого выражения служит метод setInterval( ) объекта window:

setInterval(выражение, период, [, язык])

Первым параметром является строка, например вызов функции. Период указывается в миллисекундах. Третий параметр – необязательный, в котором указывается язык с помощью которого написано заданное выражение. По умолчанию – JavaScript.

Метод setInterval( ) возвращает некоторое целое число – идентификатор временного интервала, который может быть использован в дальнейшем, например для прекращения выполнения процесса методом clearInterval( ).Например,

```
var pr = setInterval( “myfunc(), 100” )
if (confirm ( “Прервать процесс?”) )
clearInterval(pr)
```

Если требуется выполнить действие с некоторой временной задержкой, используется метод setTimeout( ), имеющий синтаксис аналогичный setInterval( ). Для отмены задержки процесса, запущенного setTimeout( ), используют clearTimeout( ).

Задание

1. Создать HTML-документ, в котором отображается список названий графических объектов и одно исходное отображение. Щелчок на элементе списка должен приводить к отображению соответствующего элемента.
2. Создать в HTML-документе две кнопки. Щелчок по кнопке ПУСК открывает через 5 секунд новое окно и загружает в него некоторый документ. Это действие может быть отменено с помощью кнопки ОТМЕНА.

Объектно-ориентированный подход. Создание классов

Класс является типом данных, определяемым пользователем. Он представляет собой модель реального объекта в виде данных и функций для работы с ними. Каждый представитель класса называется объектом.

Объединение данных и функций, которые обрабатывают объект определенного типа, в одном описании называется инкапсуляцией.

Данные класса называются полями. Функции класса – методами. Поля и методы называются элементами класса. Описание класса выглядит примерно так:

```
class <имя> {
[ private: ]
< описание скрытых элементов>
[ public: ]
< описание доступных элементов>
[ protected: ]
< описание защищенных элементов>
};
```

Описанию элементов предшествуют ключевые слова, являющиеся спецификаторами доступа.

Открытые элементы (public) доступны снаружи класса. Они, как правило, отвечают за внешний интерфейс класса и являются методами класса.

Закрытые элементы (private) предназначены только для внутреннего использования в классе. Как правило, они являются полями. Закрытые элементы доступны только методам класса, в состав которого они входят.

Защищенные элементы (protected**)** доступны для методов данного класса и классов, производных от него.

При создании класса программист самостоятельно решает, какие из членов класса будут общедоступными, а какие закрытыми. Однако большинство классов имеет типичную схему: закрытая часть содержит данные (поля), а открытая – функции для работы с этими данными (методы).

Спецификатор доступа не является обязательным, и если он не присутствует, по умолчанию элементы класса становятся закрытыми.

Рассмотрим класс, созданный для хранения даты и времени.

```
class TTime {
```

```
        private:
                int year,  month, day, hour, minute;
        public:
                  void Dispay( );
                  void SetTime(int d, int m, int y, int hr, int min);
};
```

Методы класса объявлены прототипами функций. Предположительно, они выполняют какие-то действия над полями. Их тела будут описаны позднее. Тогда их описанию должны обязательно предшествовать имя класса и операция разрешения видимости (::), сообщающая о принадлежности метода классу.

```
// Определение методов
void TTime::SetTime(int d, int m, int y, int hr, int min)  {
          day = d;          month = m;     year = y; minute = min;          hour = hr;
};
void TTime::Display( ) {
 char s[40];
printf ( "Дата: %2d. %2d. %4d  Время: %2d:%2d ",  day, month, year, hour, minute);
};
```

Тело метода может также содержаться внутри определения класса. В этом случае метод является встроенной (inline) функцией.

Большинство встроенных операций не могут применяться к классам. Нельзя использовать операцию сложения для двух объектов, или оператор = = для их сравнения. Но все эти операции применимы для полей классов. Для самих объектов справедливы следующие встроенные операции: выбор элемента (.), присваивание (=), получение адреса (&), косвенная адресация (*), и операция ->.

Для доступа к элементам объекта используются операции точка (выбора) при обращении к элементу через имя объекта и операция -> при обращении через указатель. Обращение таким способом возможно только к элементам со спецификатором доступа public. Получить или изменить значения закрытых элементов (private) можно только через обращение к соответствующим методам класса.

Использование методов класса TTime возможно в функции main( ) следующим образом.

```
  void main ( )   {
     TTime day;                  //объявление объекта day класса TTime
    day.SetTime( 24, 2, 1996, 4, 20);
// вызов метода SetTime для объекта day
    day.Display( );               // вызов метода  Display для объекта day
};
```

Язык C++ имеет две встроенных особенности при работе с классами – конструкторы и деструкторы – помогающие не забывать про инициализацию объектов и про очистку памяти после завершения работы с ними. Использование конструкторов и деструкторов необходимо во избежание массы досадных ошибок.

Конструктор – функция, автоматически вызываемая при создании объекта (инициализации класса). Внутрь конструкторов могут быть помещены процедуры инициализации для установки необходимых значений полей до использования объекта. В одном классе можно объявлять сразу несколько конструкторов. Это требуется для инициализации объектов различными способами.

Конструктор является методом класса и носит тоже имя. Он вызывается автоматически компилятором при создании каждого представителя класса. Если конструктор в программе не определен, то компилятор автоматически генерирует его без параметров (конструктор умолчанию).

Конструктор не возвращает значение, поэтому в его описании возвращаемый тип

не указывается, причем даже void. Возможно объявление несколько конструкторов с разными наборами параметров для разных видов инициализации. Конструктор, вызываемый без параметров, называется конструктором по умолчанию. Конструктор не наследуется;

Синтаксис объявления конструктора аналогичен синтаксису объявления любого другого метода класса. Например,

```
class XYValue {
int x, y;
public:
        XYValue (int X = 100, int Y = 10) {x=X;  y=Y; }
 };
```

Примерами вызова конструкторов для объектов данного класса могут служить следующие конструкции:

```
XYValue S( 200, 300), M(50), Z;
```

При использовании нескольких конструкторов в одном классе необходимо помнить, что они должны отличаться набором параметров. Каждый из них должен иметь уникальный набор параметров, иными словами, уникальную сигнатуру.

Деструктор является дополнением конструктора. По завершению работы с объектом автоматически вызывается функция, называемая деструктором.

Он также носит имя класса, но ему предшествует префикс-тильда (~). Деструктор вызывается всякий раз, когда уничтожается представитель класса. Для деструктора выполняются те же правила, что и для конструктора.

Задание

В соответствии с вариантом составить описание класса. Класс должен обязательно содержать не менее двух конструкторов – по умолчанию и с параметрами, деструктор. Объявить объекты созданного класса и продемонстрировать работу всех методов.

Варианты

1. Организовать класс *матрица,* содержащий методы вывода матрицы в общепринятом виде, нахождения транспонированной матрицы и определителя матрицы.
2. Организовать класс *треугольник,* определенный по длинам трех сторонам содержащий методы нахождения периметра и площади (по формуле Герона).
3. Организовать класс *треугольник,* определенный по координатам вершин и содержащий конструктор, деструктор, функции нахождения длин сторон, периметра, и высоты на большую сторону.
4. Организовать класс *параллелограмм*, определяемый длиной сторон и меньшим углом и содержащий методы нахождения периметра и площади параллелограмма, длин его диагоналей.
5. Организовать класс *окружность,* определяемый координатой центра и длиной радиуса. Класс должен содержать методы вычисления площади круга и длины окружности.
6. Организовать класс *дробь,* содержащий методы вывода дроби в общепринятом виде и функцию выделения целой части.

Наследование

Простое наследование описывает родство между двумя класса. Класс, являющийся прародителем называется базовым классом (родителем). Класс, созданный из базового класса называется производным классом (потомком). Производный класс наследует все элементы базового класса и может обладать новыми элементами, свойственными только ему.

На основе одного базового класса можно создавать многие классы. Производный

класс сам может быть базовым. Таким способом создается иерархия классов. Синтаксис механизма наследования:

class <идентификатор>: [ключ доступа] <идентификатор базового класса>

{тело класса};

Разница между использованием различных ключей доступа при объявлении наследования представлена в таблице:

Ключ доступа	Спецификатор в базовом классе	Ключ доступа в производном классе
private	private	нет
	protected	private
	public	private
protected	private	нет
	protected	protected
	public	protected
public	private	нет
	protected	protected
	public	public

Указание ключа доступа необязательно, по умолчанию для классов используется ключ доступа private.

```
class Base {
private:        int count;                          // поле класса
public:
                Base( ) { count = 0; }              // определение конструктора
                void SetCount(int n) { count = n; } // определение метода
                int GetCount( ) { return count; }   // определение метода
};
```

Если требуется класс, имеющий все свойства Base, но, дополнительно, обладающий способностью изменения значения поля объекта на заданную величину, можно объявить производный класс вида

```
class Derived: public Base {
public: Derived( ):Base( ){ };
void ChangeCount(int n) {SetCount( GetCount( ) + n ) };
};
```

Вызов функций базового класса предпочтительнее копирования фрагментов кода из базового класса в производный. Кроме сокращения объема программы, этим достигается упрощение ее модификации.

Задание

Изменить программу предыдущего задания, организовав на основе существующего класса производный класс. Производный класс также должен содержать не менее двух

конструкторов.

Исходная программа должна содержаться в двух файлах. В первом описание классов, во втором реализация, т.е. работа с объектами. Продемонстрировать работу всех методов базового и производного классов.

Варианты

1. Организовать производный класс, дополнительно содержащий функции нахождения произведения и сложения матриц.
2. Организовать производный класс, содержащий дополнительно функцию нахождения углов треугольника, высоты и новой функции нахождения площади по основанию и высоте.
3. Организовать производный класс, содержащий дополнительно функцию нахождения площади по высоте и основанию.
4. Описать производный класс параллелепипед, наследующий свойства базового класса и определяющий объем параллелепипеда, длины диагоналей основания и самого параллелепипеда.
5. Организовать производный класс *конус*, определенный по радиусу основания и координатой вершины, и содержащий функции нахождения площади поверхности и объема конуса.
6. Организовать производный класс, содержащий функции вычисления сложения и вычитания дробей.

Полиморфизм: виртуальные функции, перегрузка функций

Полиморфизм – это третий принцип, лежащий в основе создания классов. При полиморфизме родственные объекты (т.е. происходящие от одного базового класса) могут вести себя по-разному в зависимости от ситуации, возникающей в момент выполнения программы. Для этого один и тот же метод базового класса переопределяют в каждом классе потомке.

Виртуальные функции должны объявляться в обоих классах с атрибутом virtual, записываемым перед типом функции.

Например,

virtual void Set (int x, int y);

Правила объявления виртуальных функций:

Если в базовом классе метод определен как виртуальный, то метод, определенный в производном классе с тем же именем и набором параметров, автоматически становится виртуальным, а отличающимися параметрами - обычным.

Виртуальные методы наследуются, т.е. переопределять их в производном классе требуется только при необходимости задания различающихся действий.

Если виртуальный метод переопределен в производном классе, объекты этого класса могут получить доступ к методу базового класса с помощью операции разрешения видимости.

Виртуальный метод не может быть объявлен как static, но может быть дружественным.

По отношению ко всем виртуальным методам компилятор применяет стратегию динамического связывания. Это означает, что на этапе компиляции он не определяет, какой из методов должен быть вызван, а передает ответственность программе, которая принимает решение на этапе выполнения, когда уже известно, каков тип объекта.

Виртуальная функция вызывается только через указатель или ссылку на базовый класс. Определение того, какой экземпляр виртуальной функции вызывается, зависит от класса объекта, адресуемого указателем или ссылкой, и осуществляется во время выполнения.

Например,

```
#include <iostream.h>
class Base {
protected: int x;
public: Base (int y) {x=y;}
                    void PrintX( ) { cout<< “x=”<<x;}
                    virtual ModifyX( ) {x*=2;}
                    };
class Derived : public Base {
          public: Derived ( int f) :  Base( f ) {}
                              virtual ModifyX( ) {x/=2;}
};
int main ( )
{ Base b(10); Derived d(10);
  b.PrintX( ); d.PrintX( );
Base *pB;
pB=&b;   pB-> ModifyX( ); pB-> PrintX( );
pB=&d;   pB-> ModifyX( ); pB-> PrintX( );
}
```

Программа выведет

x=10 x=10 x=20 x=5

Если в классе вводится описание виртуального метода, он должен быть определен хотя бы как чисто виртуальный.

Виртуальная функция, объявленная в базовом классе иерархии порождения, часто никогда не используется для объектов этого класса, т.е. не имеет определения. Чтобы подчеркнуть это, используется следующая запись:

```
virtual int init(void) = 0;
```

Такие функции называются *чисто виртуальными функциями*.

Класс с одной или с большим количеством чисто виртуальных функций называется *абстрактным классом*. Абстрактный класс может быть использован только как базовый класс для последующих порождений новых классов. Следовательно, нельзя создавать объекты абстрактного класса и нельзя использовать его в качестве типа значения, возвращаемого функцией, и типа параметров функции.

Пример

```
class Shapes {
protected:      int x, y;
public:         virtual void draw ( ) = 0;
                virtual void rotate ( int ) = 0;
};
class Circle : public Shapes {
private:        radius;
public:         circle ( int r, int xx, int yy ) { r=radius; x=xx; y=yy;}
                void draw ( ) { …}
                void rotate ( int u ) { …}
};
```

Теперь невозможно создать объекта класса Shapes, поскольку он содержит чисто виртуальные функции. Объявлять объекты класса Circle возможно. Так как виртуальные функции draw ( ) и rotate ( ) в нем переопределены.

Если базовый класс содержит хотя бы один базовый метод, то рекомендуется снабжать этот класс виртуальным деструктором, даже если он ничего не делает. Это предотвратит некорректное удаление объектов производного класса, адресуемых через указатель на базовый класс.

Под перегрузкой функций (не обязательно методов класса) понимается создание нескольких прототипов функции, имеющих одинаковое имя. Компилятор различает их по набору аргументов. Перегруженные функции оказываются весьма полезными, когда одну и ту же операцию необходимо выполнить над аргументами различных типов. Какую именно функцию требуется вызвать, компилятор определяет по типу фактических параметров. Этот процесс называется разрешением перегрузки. Тип возвращаемого значения в разрешении перегрузки не участвует.

Разрешается осуществлять не только перегрузку функций, но и операторов. В большинстве языков программирования реализована концепция перегрузки операторов, пусть и в неявном виде. Так, например, оператор суммирования позволяет складывать значения разных типов.

В создаваемом классе можно изменить свойства большинства стандартных операторов, таких как +, -, *, /, заставив их работать не только с данными базовых типов, но и также с объектами.

Перегружать явным образом можно большинство операций, за исключением: .
?: # ## sizeof.

Перегрузка осуществляется с помощью методов специального типа (функций-операций). Синтаксис функции-операции:

<тип> operator <операция> (список параметров) {тело функции}

Например,

```
class angle {
                int degree, minite, second;
public:         angle_value (char *);
                int operator > ( angle &a) {   // перегруженный оператор
 if (3600*degree+60*minite+second>3600*a.degree+60*a.minite+a.second)
    return 1;
 else return 0;
  }
};
```

На перегрузку операторов накладываются следующие ограничения:

- сохраняются количество аргументов, приоритет оператора и порядок группировки его операндов, используемые в стандартных типах данных;
- невозможно изменить синтаксис оператора;
- смысл стандартного оператора применительно к базовым классам переопределять нельзя;
- невозможно создавать новые операторы,
- функции-операции не наследуются;
- функции-операции не могут быть объявлены как static.

```
#include <iostream.h>
#include <string.h>
#include <conio.h>
const size = 80;
class Phrase {
private:        char str[size];
public:
        Phrase ( ) { strcpy (str, " "); }
        Phrase (char *s) { strcpy (str, s); }
        void display ( ) { cout<< str<<endl;}
        Phrase operator + = (Phrase a)              // перегрузка оператора + =
        {       if ( strlen(str) + strlen(a. str) < size) strcat (str, a. str);
                else cout<< "строка слишком длинная";
                return (*this);
```

```
        }
};
void main ( )
{ clrscr( );
Phrase b;  Phrase c ( " тест");
b+=c;     b. display( );
Phrase d ( "еще ");
d + = c+ =d;  d. display( );
}
```

Варианты

1. Изменить программу предыдущей лабораторной работы, используя для сложения и умножения матриц перегрузку соответствующих операций.

2.Изменить программу предыдущей лабораторной работы, определив виртуальную функцию для нахождения площади таким образом, чтобы в базовом и производном классе тела функций определялись по-разному.

3.Описать абстрактный класс *Животное*. Класс должен содержать характеристики животных: название, вид, местообитание, функцию вывода всех данных на экран. На его основе реализовать классы *Млекопитающее*, *Рыба*, *Птица*. Отдельными характеристиками классов являются: для млекопитающих – травоядное, хищник или всеядное; для рыб – морская или пресноводная; для птиц – дикая, домашняя, если дикая перелетная, или нет.

4.Создать абстрактный класс *Средство передвижения*. На его основе реализовать классы *самолет*, *машина*, *корабль*. Все классы должны хранить параметры средств передвижения: скорость, расход топлива, наименование производителя, год выпуска, метод вывода на экран всех данных, определения срока службы. Индивидуально для самолета указать высоту и максимальную дальность полета, для машины – объем двигателя, для самолета и корабля – количество посадочных мест, для корабля – водоизмещение.

5. Создать абстрактный класс правильный многоугольник. На его основе создать классы треугольник, квадрат, пятиугольник. Предусмотреть методы создания объектов, вычисления их периметра, площади, величины угла.

6.Изменить программу предыдущей лабораторной работы, используя для вычитания и сложения дробей соответствующие перегруженные операции.

2. Доказательство правильности программ

При доказательном подходе разработка алгоритмов и программ предполагает составление спецификаций и доказательство их правильности по отношению к этим спецификациям. Процесс разработки программ считается завершенным после проверки их на ЭВМ и предоставлении доказательств отсутствия ошибок.

Доказательства правильности алгоритмов и программ, равно как и любые другие доказательства, строятся на основе суждений и рассуждений. В данном случае суждения и рассуждения касаются результатов выполнения алгоритмов и программ с теми или иными данными.

Конструктивно, доказательства правильности алгоритмов и программ строятся на суждениях и утверждениях о результатах выполнения каждого из составляющих их действий и операций в соответствии с порядком их выполнения.

В качестве примера проведем анализ результатов алгоритма, состоящего из трех присваиваний.

алг «$y = x^5$»	Результаты	Утверждения
нач		
v := x		

×x	v1 = x×x	Þ	$v1 = x^2$
v :=			
v			
×			
v	v2 = v1×v1	Þ	$v2 = x^4$
y :=			
v			
×			
x	y = v2×x	Þ	$y = x^5$
кон			

Справа от алгоритма приведены результаты выполнения присваиваний. Результатом первого присваивания v := x×x будет значение v1 = x×x переменной v. Результат следующего присваивания v := v×v - второе значение переменной v, равное v2 = v1×v1 . Результатом третьего присваивания y := v×x будет значение y = v2×x .

На основе приведенных рассуждений, можно сделать три утверждения о промежуточных и конечных результатах вычислений:

Результаты		Утверждения
{ v1 = x×x	Þ	$v1 = x^2$
{ v2 = v1×v1	Þ	$v2 = x^4$
{ y = v2×x	Þ	$y = x^5$

Таким образом, можно высказать окончательное утверждение. Конечным результатом выполнения будет $y = x^5$ для любых значений x.

Доказательство. Исходя из описания результатов выполнения присваиваний значение y будет равно y = v2×x = (v1×v,)×x = ((x×x).(x×x))) ×x = x^5.

Что и требовалось доказать.

Техника анализа и доказательства правильности алгоритмов и программ во многом совпадает с техникой доказательства любых других утверждений и состоит в применении следующих четырех приемов: разбор случаев; подбор контрпримеров; выделение лемм; индуктивный вывод.

Разбор случаев применяется для анализа результатов выполнения конструкций альтернативного выбора. В качестве примера проведем анализ приведенного выше алгоритма «выбора» максимума трех чисел, содержащего выбор альтернатив.

алг «y = max(a, b,c)»	Результаты
нач	
если a > b то	при a > b
y = a	y = a
ин ес b > c то	при b > c
y = b	y = b
ин ес c > a то	при c > a
y = c	y = c
к если	при не (b > c)
кон	

Справа от алгоритма приведены результаты вычислений с указанием условий, при которых они получаются. На основании этих фактов можно заключить, что конечные результаты вычисления имеют три варианта:

a, при a > b,
y = b, при b > c и b ³ a,
c, при c > a и c ³ b.

В то же время значение максимума должно быть равно:

a, при a ³ b и a ³ c,
max = b, при b ³ c и b ³ a,
c, при c ³ a и c ³ b.

Во всех трех случаях видны различия в условиях получения и определения максимальных значений. Покажем, что эти различия существенны и утверждение о том, что алгоритм дает правильные результаты для всех данных, неверно.

Для опровержения общего утверждения достаточно указать хотя бы один контрпример. В данном случае утверждение о правильности алгоритма гласило бы: для любых значений переменных a, b, c конечным было бы значение max (a, b, c).

Контрпримером в данном случае будут значения: a = 2, b = 1, c = 3. Для этих данных по определению max = 3, а по результатам выполнения алгоритма y = 2. Следовательно, в алгоритме содержится ошибка.

Постановка задачи Метод решения

Вычисление max (a, b, c).

Дано: a, b, c - три числа, mx = max(max(a,b),c)
Требуется: mx - максимум, max(x,y) = x, при x ³ y
где: mx = max (a, b, c). y, при y ³ x

Данный метод решения непосредственно состоит из формул определения максимумов из трех и двух чисел. Реализация этого метода в форме алгоритма может быть такой:

алг «тх = тах(a,b,c)»	Результаты
нач	
если a ³ b то	при a ³ b
тх = a	mx = a
иначе	при b > a
mx = b	mx = b
кесли { mx = max(a,b) }	при c < mx
если c ³ mx то	при c ³ mx
mx = c	mx' = c
кесли	
кон	

Доказательство правильности алгоритмов можно проводить двумя способами. Первый способ - анализ правильности при построении алгоритмов. Второй способ - анализ правильности после построения алгоритмов.

Первый способ - показать, что алгоритм является корректной реализацией метода решения, и доказать, что метод - правильный. Для рассмотренного алгоритма это доказательство изложено выше.

Привлечение для создания алгоритмов известных методов решения, для которых доказана их правильность, позволяет существенно упростить обоснование правильности программ. При этом центр тяжести проблем смещается к созданию и обоснованию гарантированно правильных методов решения задач.

Второй способ - исчерпывающий анализ результатов выполнения алгоритма на соответствие постановке решаемых задач для любых допустимых данных. Это - доказательство путем исчерпывающего анализа результатов выполнения алгоритмов и программ.

Результаты выполнения рассматриваемого алгоритма вычисления максимума трех чисел приведены справа от него. Анализ результатов алгоритмов, содержащих конструкцию выбора, требует разбора случаев. Отметим, что все эти случаи были уже указаны ранее при разборе ошибочной версии алгоритма.

Для обоснования правильности алгоритма докажем вспомогательное утверждение о результатах выполнения конструкции альтернативного выбора

Лемма: Конечными результатами выполнения алгоритма

Алгоритм	Результаты
если a > b то	при a ³ b
тх := a	mx = a
иначе	при b > a
тх := b	mx = b
кесли	

является значение mx = max(a, b) для любых значений a и b.

Доказательство. Результатом вычислений здесь будут значения a при a ³ b mx = b при a < b, что совпадает с определением max (a, b).

С помощью этой леммы легко доказать правильность алгоритма в целом.

{ mx = max (a, b) }	Результаты
если c ³ mx то	при c ³ mx
mx := c	mx' = c
кесли	
	mx' = mx
при c < mx	

Утверждение. Конечным результатом выполнения алгоритма вычисления максимума будет значение mx' = max (a, b, c) для любых значений a, b и c.

Доказательство лемм - основной прием доказательства правильности сложных алгоритмов и программ. Напомним, что лемма — это вспомогательное утверждение, предполагающее отдельное доказательство.

Одним из важнейших применений аппарата лемм является анализ результатов выполнения и доказательство правильности алгоритмов с циклами. Используемые для анализа циклов леммы называются индуктивными утверждениями. Эти леммы выражают утверждения о промежуточных результатах выполнения циклов.

В качестве примера использования индуктивных рассуждений рассмотрим алгоритм вычисления среднего арифметического последовательности чисел. В приводимом алгоритме предполагается, что последовательность чисел размещена в массиве X[1:N].

алг «среднее значение»	
массив	
X[1:N]	
нач	Результаты:
от	
k = 1 до N цикл	
S :=	
S *	
(k-l)/k +	
X[k]/k	Sk = Sk-1*(k-l)/k + X[k]/k
кцикл	[k = (1...N)]
Xср :=	

S Xcp = S

кон

Этот алгоритм обычно считается ошибочным. «Ошибкой» в этом алгоритме считается отсутствие присваивания S := 0 перед началом цикла.

Разберем результаты выполнения алгоритма на первых трех шагах:

S1 = S0×(l - 1)/1 + X[1]/1 = S0×0/1 + X[1]/1 = X[1]/1;

S2 = S1×(2 - 1)/2 + X[2]/2 = S1×1/2 + X[2]/2 = X[1]/2 + X[2]/2;

S3 = S2×(3 - 1)/3 + X[3]/3 = S2×2/3 + X[3]/3 = X[1]/3 + X[2]/3 + X[3]/3.

Можно утверждать, что на первых трех шагах результатом является среднее арифметическое обрабатываемых чисел. На основе этих примеров можно сделать индуктивное утверждение - «на каждом очередном k-м шаге выполнения цикла результатом будет среднее арифметическое»

Sk = Sk-1×(k-l)/k + X[k]/k = X[l]/k + X[2]/k + ... + X[k]/k.

Доказательство этого утверждения проводится с помощью математической индукции. На первом шаге при k = 1 оно уже доказано. Допустим, что оно справедливо на (k -1)-м шаге

Sk-1 = X[l]/(k-l) + X[2]/(k-l) + ... + X[k-l]/(k-l).

Подставим его в описание результатов цикла на k-м шаге

Sk= Sk-1×(k-l)/k +X[k]/k.

Тогда результат выполнения цикла на k-м шаге оказывается равным

Sk = X[l]/k + X[2]/k + ... + X[k-l]/k + X[k]/k,

т. е. среднему арифметическому первых k чисел.

Таким образом, индуктивное утверждение доказано. В силу математической индукции это утверждение верно для всех k = l, 2, ..., N. Следовательно, на последнем шаге конечным результатом выполнения цикла станет значение

SN = SN-1×(N-1) + X[N]/N = X[1]/N + ... + X[N]/N.

Исходя из этого утверждения конечным результатом выполнения алгоритма в целом будет среднее арифметическое значение

Xcp = SN = X[1]/N + ... + X[N]/N.

Следовательно, приведенный алгоритм, несмотря на содержащуюся в нем «ошибку», является правильным. В целом анализ правильности алгоритмов с циклами во многом построен на использовании индукции.

Индукция - это вывод общих суждений из частных примеров. При анализе циклов она используется для подбора индуктивных утверждений о промежуточных результатах выполнения циклов. Однако для доказательства правильности индуктивных утверждений о результатах выполнения циклов используется полная математическая индукция.

Математическая индукция - это принцип доказательства последовательностей утверждений P(1), P(2), P(3), ..., P(N), ... когда известно, что верны первые утверждения для n = 1, 2, 3 и из истинности (n - 1)-го утверждения следует истинность n-го утверждения:
Принцип математической индукции: если первое утверждение P(1) истинно и из утверждения P(n - 1) следует утверждение P(n), то истинны все утверждения P(1), P(2), P(3), ..., P(n), ...

Приведем примеры индуктивного анализа циклов для алгоритма нахождения минимального значения в последовательности чисел, который в этот раз действительно будет ошибочным.

алг «нахождение минимума»

массив

x[1:N]

нач Результаты:

```
от k = 1 до N цикл
если
 x[k] < min то
тп :=
 x[k]                          mnk = { x[k], при x[k] < mnk-1,
все                                  { mnk-1, в ином случае
кцикл                                  [ k = (1 ... N)]
Min := тп                                 Min = mnN
кон
```

Утверждение. Приведенный алгоритм определения минимального значения последовательности чисел неправильный.

Доказательство. Для демонстрации ошибок необходимо привести контрпример. Для построения контрпримера разберем результаты выполнения на первом шаге цикла.

Результаты выполнения первого шага цикла при k = 1:

x[1] при x[1] < mn0

mn1 = min (x[1], mn0).

mn0 при x[1] £ mn0

Следовательно, результатом будет mn1 = min (x[l], mn0)

Однако поскольку начальное значение mn0 неизвестно, то неопределено значение результата выполнения первого шага цикла. Аналогичное утверждение можно сделать о втором и всех последующих шагах выполнения цикла:

mnk = min (x[k], Min(x[k-l], ..., x[1], mn0) = Min (x[k], x[k-1], ..., x[1], mn0).

В силу математической индукции это утверждение справедливо при k = N:

mnN = Min (x[N], x[N - 1], ..., x[2], x[1], mn0),

Таким образом на основании этого утверждения можно сделать заключение о конечном результате выполнения алгоритма в целом:

Min = mnN = Min (x[N], x[N - 1], ..., x[2], x[1], mn0).

Из этой формулы видно, что конечный результат равно как и результат первого присваивания зависит от начального значения mn0 переменной mn. Однако эта величина не имеет определенного значения, соответственно неопределен конечный результат выполнения алгоритма в целом, что и является ошибкой.

В самом деле, если значение mn0 окажется меньше любого из значений последовательности x[1], ... x[N], то конечный результат вычислений будет неправильным. В частности, при реализации алгоритма на Бейсике неправильный результат будет получен, если последовательность будет состоять только из положительных чисел. Например, для последовательности чисел: 1, 2, 3, ..., N.

Приведем правильную версию алгоритма с доказательством отсутствия ошибок, используя технику индуктивных утверждений.

```
алг «нахождение минимума»
массив x[1:n]
нач
  тп := x[1]
 от k = 1 до N цикл
   если
 x[k] < тп то
    тп =
 x[k]
   все
  кцикл

  Min = тп
```

кон
Результаты:
mn0 = x[1]
[k = (1 ... N)]
Min = mnN

Утверждение. Для любой последовательности чисел x[l:N] конечным результатом вычислений будет значение Min = Min (x[1], ..., x[N]).

Доказательство. Воспользуемся результатами анализа выполнения алгоритма, рассмотренного ранее. Различие между ними состоит в добавлении перед началом цикла присваивания mn := x[1], которое задает начальное значение переменной mn, равное mn0 = x[1].

Тогда в силу приведенных ранее рассуждений и выкладок конечным результатом выполнения новой версии алгоритма будет значение

Min = mnN = Min(x[N], x[N-l], ..., x[2], x[1], mn0) = Min(x[N], x[N-l], ..., x[2], x[l], x[l]) = Min(x[N], ... x[1]).

Рассмотренные примеры являются образцами доказательств правильности алгоритмов и программ, которые могут использоваться для анализа и доказательства правильности других новых алгоритмов и программ обработки данных.

Общий вывод, который мы хотим сделать, состоит в том, что применение доказательных методов превращает программирование в научную дисциплину создания безошибочных алгоритмов и надежных программ для ЭВМ.

Контрольные вопросы:

1. Как показать наличие ошибок в алгоритме?
2. Сколь долго может продолжаться отладка программ?
3. Зачем нужны доказательства в анализе алгоритмов?
4. Из чего состоит техника доказательств правильности?
5. Когда применяется разбор случаев?
6. Что такое леммы?
7. Что такое индуктивные рассуждения?

Задание

1. Приведите постановку, алгоритм решения и разбор правильности для следующих задач:

а) подсчет суммы целых чисел;
б) подсчет суммы нечетных чисел;
в) подсчет членов арифметической прогрессии;
г) подсчет членов геометрической прогрессии.

2. Для последовательности чисел x_1, x_2,..., x_N, приведите постановку, алгоритм решения и разбор правильности следующих задач:

а) подсчет суммы всех чисел;
б) вычисление среднего арифметического чисел;
в) определение наибольшего из чисел;
г) определение наименьшего из чисел.

3. Для данных о росте и весе учеников приведите постановку задачи, алгоритм решения и разбор правильности для следующих задач:

а) нахождение самого высокого ученика;
б) нахождение самого легкого ученика;
в) нахождение среднего роста учеников;

г) нахождение суммарного веса учеников.

4. Для прямоугольной матрицы A размерности nxm приведите постановку, алгоритм решения и разбор правильности следующих задач:

а) подсчет сумм элементов матрицы по столбцам;

б) нахождение минимального значения в каждом столбце;

в) нахождение максимального значения в каждой строке;

г) нахождение наибольшего из минимальных значений в столбцах;

д) нахождение наименьшего из максимальных значений в строках.

5. Для N точек на плоскости, заданных случайным образом, приведите постановку, метод решения, сценарий, алгоритм и программу решения следующих задач:

а) найти точку, наиболее удаленную от центра координат;

б) соединить пару наиболее удаленных точек;

в) найти три точки, образующие треугольник с наибольшим периметром;

г) найти три точки, образующие треугольник с наибольшей площадью.

3. Технология написания тестов

К технологиям написания тестов относятся:

1. Smoke Test - тестирование выпущенного приложения для принятия\непринятие на стадию дальнейшего тестирования (обычно представляет собой функциональную проверку без излишних премудростей аля "лишь бы делало то, что должно делать"). Для этого вида теста сценарии пишутся как можно короче и не очень подробно, в основном покрывая необходимый минимум функциональности...

2. Critical Path Test - полный функциональный тест приложения, при условии, что оно прошло Smoke Test. Особенно это актуально, если времени на тестирование приложения отведено мало. Для данного вида тестирования пишутся наиболее подробные и глубокие тест-кейсы дабы покрыть всю возможную функциональность приложения.

3. Extended Test - тестируются нестандартные ситуации в приложении, например, ввод спец-символов, нелогичное нажатие кнопок, открыть одно окно и закрыть предыдущее ну и так далее.

4. User Acceptance Test - данный вид тестирования является сборным из выше описанных технологий и представляет собой те тесты, которые с наибольшей вероятностью повторит конечный пользователь.

Задание

Продемонстрируйте создание всех изложенных видов тестирования любой из выполненных лабораторных работ.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО САМОСТОЯТЕЛЬНОЙ РАБОТЕ СТУДЕНТОВ

Самостоятельная работа – планируемая учебная, учебно-исследовательская, научно-исследовательская работа студентов, выполняемая во внеаудиторное время по заданию и при методическом руководстве преподавателя, но без его непосредственного участия (при частичном непосредственном участии преподавателя, оставляющем ведущую роль за работой студентов).

Количество часов самостоятельной работы и ее распределение между дидактическими единицами приведено в рабочей программе.

Задачами самостоятельной работы являются:

систематизация и закрепление полученных теоретических знаний и практических умений студентов;

углубление и расширение теоретических знаний;

формирование умений использовать нормативную, правовую, справочную документацию и специальную литературу;

развитие познавательных способностей и активности студентов: творческой инициативы, самостоятельности, ответственности и организованности;

формирование самостоятельности мышления, способностей к саморазвитию, самосовершенствованию и самореализации;

развитие исследовательских умений;

использование собранного и полученного в ходе самостоятельных занятий для эффективной подготовки к итоговым зачетам и экзаменам.

В процессе самостоятельной работы студент приобретает навыки самоорганизации, самоконтроля, самоуправления, становится активным самостоятельным субъектом учебной деятельности.

Выполняя самостоятельную работу под контролем преподавателя, студент должен:

– освоить минимум содержания, выносимый на самостоятельную работу студентов и предложенный преподавателем в соответствии с Федеральными государственными образовательными стандартами высшего профессионального образования по данной дисциплине.

– планировать самостоятельную работу в соответствии с графиком самостоятельной работы, предложенным преподавателем.

– самостоятельную работу студент должен осуществлять в организационных формах, предусмотренных учебным планом и рабочей программой преподавателя.

– выполнять самостоятельную работу и отчитываться по ее результатам в соответствии с графиком представления результатов, видами и сроками отчетности по самостоятельной работе студентов.

Основной формой самостоятельной работы студента является изучение литературы, рекомендованной в рабочей программе, подготовка доклада и презентации по выбранной тематике, представление доклада на практическом занятии, подготовка отчетов к лабораторным работам.

КОНТРОЛЬ ЗНАНИЙ

Текущий контроль знаний предполагает защиту лабораторных работ и выполнение тестовых заданий. В тестовые задания входят вопросы, рассматриваемые на практических занятиях, примеры вопросов приведены в рабочей программе. По окончанию изучения дисциплины сдается зачет. Вопросы к зачету приведены в рабочей программе.

СОДЕРЖАНИЕ