Учреждение образования
«Белорусский государственный университет культуры и искусств»

Факультет информационно-документных коммуникаций
Кафедра информационных ресурсов и коммуникаций

СОГЛАСОВАНО
Заведующий кафедрой

_____ Ж. Л. Романова
___ _____ 2020 г.

СОГЛАСОВАНО
Декан факультета

_____ Ю. Н. Галковская
___ _____ 2020 г.

УЧЕБНО-МЕТОДИЧЕСКИЙ КОМПЛЕКС
ПО УЧЕБНОЙ ДИСЦИПЛИНЕ

АЛГОРИТМИЗАЦИЯ И ОСНОВЫ ПРОГРАММИРОВАНИЯ

для специальности 1-23 01 11 Библиотечно-информационная деятельность
(по направлениям),
направления специальности 1-23 01 11-02
Библиотечно-информационная деятельность (автоматизация)

Составители:
Е. Э. Политевич, доцент кафедры информационных ресурсов и коммуникаций, кандидат педагогических наук
Е. А. Шишкова, старший преподаватель кафедры информационных ресурсов и коммуникаций, магистр педагогических наук

Рассмотрено и утверждено
на заседании Совета университета 17 ноября 2020 г.
протокол № 2

Составители:

Е. Э. Политевич, доцент кафедры информационных ресурсов и коммуникаций учреждения образования «Белорусский государственный университет культуры и искусств», кандидат педагогических наук;

Е. А. Шишкова, старший преподаватель кафедры информационных ресурсов и коммуникаций учреждения образования «Белорусский государственный университет культуры и искусств», магистр педагогических наук

Рецензенты:

П. В. Гляков, профессор кафедры информационных технологий в культуре учреждения образования «Белорусский государственный университет культуры и искусств», кандидат физико-математических наук, доцент;

Ученый совет ГУ «Белорусская сельскохозяйственная библиотека им. И.С. Лупиновича Национальной академии наук Беларуси»

Рассмотрен и рекомендован к утверждению:

Кафедрой информационных ресурсов и коммуникаций учреждения образования «Белорусский государственный университет культуры и искусств» (протокол от 26.12.2019 № 5);

Советом факультета информационно-документных коммуникаций учреждения образования «Белорусский государственный университет культуры и искусств» (протокол от 26.05.2020 № 9)

СОДЕРЖАНИЕ

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Электронный учебно-методический комплекс по учебной дисциплине «Алгоритмизация и основы программирования» предназначен для сопровождения образовательной деятельности студентов при изучении названной дисциплины. Его структура и наполнение соответствуют Положению об учебно-методическом комплексе по учебной дисциплине, утвержденному приказом ректора университета от 26.04.2017 № 69. Электронный учебно-методический комплекс предназначен для помощи студентам в овладении знаниями, умениями и навыками создания алгоритмов и программ для автоматизации обработки информации с помощью ЭВМ, а также получения опыта использования компьютерных информационных технологий.

Актуальность учебной дисциплины «Алгоритмизация и основы программирования» обусловлена необходимостью подготовки специалистов библиотечно-информационной сферы, обладающих знаниями, умениями и навыками в области автоматизации библиотечно-библиографических процессов, готовых к использованию передовых методов и средств алгоритмизации и программирования библиотечно-библиографических и информационных процессов в профессиональной деятельности.

Актуальной задачей для специалистов в сфере библиотечно-информационной деятельности является содержательное и деятельностное включение знаний по основам алгоритмизации и программирования в систему профессиональных компетенций библиотекарей-библиографов, что требует специальной профессиональной подготовки. Учебная дисциплина «Алгоритмизация и основы программирования» способствует получению знаний о различных формах организации данных в программах и методах их обработки при решении практических задач автоматизации библиотечно-информационной деятельности, умений использовать практические приемы, методы и средства анализа, построения и использования веб-программирования в библиотечно-информационной деятельности, а также формирование практических навыков разработки алгоритмов и программ.

Целью электронного учебно-методического комплекса по учебной дисциплине «Алгоритмизация и основы программирования» является систематизация учебно-методических материалов, необходимых для овладения студентами теоретическими и практическими основами алгоритмизации и программирования в библиотечно-информационной деятельности; учебно-методическая помощь студентам в приобретении умений и навыков разработки блок-схем алгоритмов, решения задач автоматизации библиотечно-информационной деятельности, форм ввода и

вывода информации в веб-приложениях, создания программы на языке программирования JavaScript для обработки информации на стороне клиента, получения опыта владения методикой использования программных средств для решения практических задач при осуществлении библиотечно-информационной деятельности.

Целевая направленность электронного учебно-методического комплекса обусловила необходимость решения ряда следующих *задач:*

– систематизация нормативно-правовой, научно-практической, учебно-методической информации, отражающей проблемное поле учебной дисциплины;

– упорядочение процесса изучения учебной дисциплины с учетом требований, предъявляемых педагогикой высшей школы к лекциям, практическим, лабораторным и семинарским занятиям;

– формирование на основе междисциплинарного подхода системных знаний о методах, средствах и технологиях алгоритмизации и программирования библиотечно-библиографических и информационных процессов;

– развитие навыков использования инструментальных средств программирования библиотечно-библиографических и информационных процессов;

– оказание методической помощи студентам в освоении учебного материала;

– организация контролируемой самостоятельной работы и контроля знаний студентов;

– методическое и информационное обеспечение учебной дисциплины и контролируемой самостоятельной работы студентов;

– развитие способностей к постоянному самообразованию, в том числе в отношении профессионально значимых знаний в области алгоритмизации и программирования, а также к эффективной самореализации в профессии.

Электронный учебно-методический комплекс по учебной дисциплине «Алгоритмизация и основы программирования» состоит из четырех разделов. В *теоретическом разделе* размещен конспект лекций. Материал структурирован по темам в соответствии с учебной программой дисциплины. *Практический раздел* содержит методические указания к лабораторным и семинарским занятиям, а также материал для их проведения: тематику и описание практических и лабораторных работ, тематику семинарских занятий, вопросы и литературу, рекомендуемую к изучению. В *разделе контроля знаний* электронного учебно-методического комплекса представлены методические указания к контролируемой самостоятельной работе студентов, ее тематика и задания с инструкциями к выполнению

самостоятельной работы, вопросы к зачету и экзамену, описаны рекомендуемый диагностический инструментарий для оценки учебных достижений студентов и критерии оценки знаний студентов. *Вспомогательный раздел* содержит учебную программу, перечень учебных изданий и информационно-аналитического материала, рекомендованных для изучения учебной дисциплины (список основной и дополнительной литературы).

В результате изучения учебной дисциплины «Алгоритмизация и основы программирования» при использовании эффективной педагогической методики и современных информационных технологий студенты должны овладеть знаниями, умениями, навыками и опытом, необходимыми для решения разнообразных профессиональных задач, и предусмотренными образовательным стандартом высшего образования ОСВО 1-23 01 11-2014 по специальности 1–23 01 11 Библиотечно-информационная деятельность (по направлениям).

ТЕОРЕТИЧЕСКИЙ РАЗДЕЛ

КОНСПЕКТ ЛЕКЦИЙ

Основными документными источниками для составления лекций по дисциплине «Алгоритмизация и основы программирования» являются:

1. Голицына, О. Л. Основы алгоритмизации и программирования : учеб. пособие для СПО / О. Л. Голицына, И. И. Попов. – 3-е изд., испр. и доп. – М. : Форум, 2008. – 432 с.

2. Жданова, Т. А. Основы алгоритмизации и программирования : учеб. пособие / Т. А. Жданова, Ю. С. Бузыкова. – Хабаровск : Изд-во Тихоокеан. гос. ун-та, 2011. – 56 с. – Режим доступа: http://window.edu.ru/resource/402/77402. – Дата доступа: 06.11.2019.

3. Захаркина, В. В. JavaScript. Основы клиентского программирования : учеб. пособие / В. В. Захаркина. – СПб. : Ф-т филологии и искусств СПбГУ, 2007. – 73 с. – Режим доступа: http://window.edu.ru/resource/394/57394. – Дата доступа: 06.11.2019.

4. Зудилова, Т. В. Web-программирование: JavaScript : учеб. пособие / Т. В. Зудилова, М. Л. Буркова. – СПб.: НИУ ИТМО, 2012. – 68 с. – Режим доступа: http://window.edu.ru/resource/612/76612. – Дата доступа: 06.12.2019.

5. Лесневский, А. С. Объектно-ориентированное программирование для начинающих / А. С. Лесневский. – М. : Бином. Лаборатория знаний, 2009. – 212 с.

6. Основы программирования : учеб. пособие / В. В. Борисенко ; Интернет-университет информационных технологий, [МГУ им. М. В. Ломоносова]. – М. : ИНТУИТ.ру, 2005. – 314 с.

7. Савельева, Н. В. Основы программирования на PHP : курс лекций : учеб. пособие : для высших учебных заведений по специальностям в области информационных технологий / Н. В. Савельева ; Интернет университет информационных технологий. – М. : ИНТУИТ.ру, 2005. – 260 с.

Тема 1. Основы теории алгоритмов

Слово «Алгоритм» происходит от algorithmi – латинского написания имени аль-Хорезми, под которым в средневековой Европе знали величайшего математика из Хорезма (город в современном Узбекистане) Мухаммеда бен Мусу, жившего в 783-850 гг. В своей книге «Об индийском счете» он сформулировал правила записи натуральных чисел с помощью арабских цифр и правила действий над ними столбиком. В дальнейшем алгоритмом стали называть точное предписание, определяющее последовательность действий, обеспечивающую получение требуемого результата из исходных данных. Алгоритм может быть предназначен для выполнения его человеком или автоматическим устройством. Создание алгоритма, пусть даже самого простого, - процесс творческий. Он доступен исключительно живым существам, а долгое время считалось, что только человеку. Другое дело – реализация уже имеющегося алгоритма. Ее можно поручить субъекту или объекту, который не обязан вникать в существо дела, а возможно, и не способен его понять. Такой субъект или объект принято называть формальным исполнителем.

Примером формального исполнителя может служить стиральная машина-автомат, которая неукоснительно исполняет предписанные ей действия, даже если вы забыли положить в нее порошок. Человек тоже может выступать в роли формального исполнителя, но в первую очередь формальными исполнителями являются различные автоматические устройства, и компьютер в том числе.

Каждый алгоритм создается в расчете на вполне конкретного исполнителя. Те действия, которые может совершать исполнитель, называются его допустимыми действиями. Совокупность допустимых действий образует систему команд исполнителя. Алгоритм должен содержать только те действия, которые допустимы для данного исполнителя.

Свойства алгоритмов:

Дискретность – значения новых величин (данных) вычисляются по определенным правилам из других величин с уже известными значениями.

Определенность (детерминированность) – каждое правило из набора однозначно, а сами данные однозначно связаны между собой, т.е. последовательность действий алгоритма строго и точно определена.

Результативность (конечность) – алгоритм решает поставленную задачу за конечное число шагов.

Массовость – алгоритм разрабатывается так, чтобы его можно было применить для целого класса задач, например, алгоритм вычисления определенных интегралов с заданной точностью.

Способы описания алгоритмов: словесная запись алгоритма, блок-схема (схема графических символов), алгоритмические языки. Существует несколько способов описания алгоритмов. Наиболее распространенные способы – это словесное и графическое описания алгоритма.

Словесное описание алгоритма

В любом алгоритме для обозначения данных используют некоторый набор символов, называемых *буквами*. Конечную совокупность букв называют *алфавитом*, из любой конечной последовательности которого можно составить *слово*, т.е. в любом алфавите реальным данным можно сопоставить некоторые слова, в дальнейшем обозначающие эти данные.

При словесной записи алгоритм описывается с помощью естественного языка с использованием следующих конструкций:

1) шаг (этап) обработки (вычисления) значений данных – «=»;

2) проверка логического условия: если (условие) истинно, то выполнить действие 1, иначе – действие 2;

3) переход (передача управления) к определенному шагу (этапу) *N*.

Для примера рассмотрим алгоритм решения квадратного уравнения вида $a \cdot x^2 + b \cdot x + c = 0$:

1) ввод исходных данных a, b, c ($a,b,c \neq 0$);

2) вычислить дискриминант $D = b^2 - 4 \cdot a \cdot c$;

3) если $D < 0$, то перейти к п. 6, сообщив, что действительных корней нет;

4) иначе, если $D \geq 0$, вычислить $x_1 = (-b + \sqrt{D})/(2 \cdot a)$ и $x_2 = (-b - \sqrt{D})/(2 \cdot a)$;

5) вывести результаты x_1 и x_2 ;

6) конец.

Графическое описание алгоритма

Графическое изображение алгоритма – это представление его в виде схемы, состоящей из последовательности блоков (геометрических фигур), каждый из которых отображает содержание очередного шага алгоритма. А внутри фигур кратко записывают действие, выполняемое в этом блоке. Такую схему называют блок-схемой или структурной схемой алгоритма, или просто схемой алгоритма.

Правила изображения фигур сведены в единую систему программной документации (дата введения последнего стандарта ГОСТ 19.701.90 – 01.01.1992).

По данному ГОСТу графическое изображение алгоритма – это схема данных, которая отображает путь данных при решении задачи и определяет этапы их обработки.

Схема данных состоит из следующих элементов:

– символов данных (символы данных могут отображать вид носителя данных);

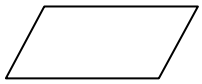
– символов процесса, который нужно выполнить над данными;

– символов линий, указывающих потоки данных между процессами и носителями данных;

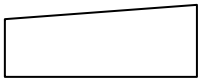
– специальных символов, которые используют для облегчения чтения схемы алгоритма.

Рассмотрим основные символы для изображения схемы алгоритма.

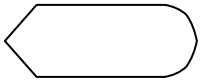
***Символы ввода-вывода данных**:*



– данные ввода-вывода, если носитель не определен;

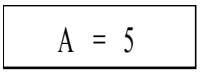


– ручной ввод с устройства любого типа, например, с клавиатуры;

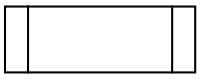


– отображение данных в удобочитаемой форме на устройстве, например, дисплее.

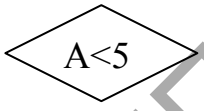
***Символы процесса**:*



– ***процесс*** – отображение функции обработки данных, т.е. операции, приводящей к изменению указанного значения;



– ***предопределенный процесс*** – отображение группы операций, которые определены в другом месте, например в подпрограмме (функции);



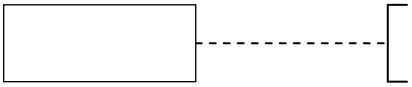
– ***решение*** – отображение функции, имеющей один вход и ряд альтернативных выходов, из которых только один может быть активизирован после анализа условия, указанного внутри этого символа.

Символы линий – отображают поток данных или управления. Линии – горизонтальные или вертикальные, имеющие только прямой угол перегиба. Стрелки – указатели направления не ставятся, если управление идет сверху вниз или слева направо.

Специальные символы

Соединитель – используется при обрыве линии и продолжении ее в другом месте (необходимо присвоить название).

Терминатор – вход из внешней среды или выход во внешнюю среду (начало или конец схемы программы).



Комментарий.

Порядок выполнения алгоритма. Основные виды алгоритмов: линейные, разветвленные и циклические. Порядок выполнения алгоритма:

1. Действия в алгоритме выполняются в порядке их записи
2. Нельзя менять местами никакие два действия алгоритма
3. Нельзя не закончив одного действия переходить к следующему

Любую программу можно разбить на блоки, реализованные в виде алгоритмов (процессов), которые можно разделить на три вида:

1) линейные (единственное направление выполнения);

2) разветвляющиеся (направление выполнения определяет условие);

3) циклические (отдельные участки вычислений выполняются многократно).

Любой циклический процесс включает в себя участок с разветвлением и может быть простым и сложным (вложенным).

Для решения вопроса о том, сколько раз нужно выполнить цикл, используется анализ переменной, которую называют параметром цикла.

Циклический процесс, в котором количество повторений заранее известно, называется циклом по счетчику, а циклический процесс, в котором количество повторений заранее неизвестно и зависит от получаемого в ходе вычислений результата, называют итерационным.

Пример простейшего линейного процесса

Наиболее часто в практике программирования требуется организовать расчет некоторого арифметического выражения при различных исходных данных. Например, такого:

$$z = \frac{tg^2 x}{\sqrt{x^2 + m^2}} + x^{(m+1)} \sqrt{x^2 + m^2} \quad ,$$

где $x > 0$ – вещественное, m – целое.

Разработка алгоритма обычно начинается с составления схемы. Продумывается оптимальная последовательность вычислений, при которой, например, отсутствуют повторения. При написании алгоритма рекомендуется переменным присваивать те же имена, которые фигурируют в заданном арифметическом выражении либо иллюстрируют их смысл.

Для того чтобы не было «длинных» операторов, исходное выражение полезно разбить на ряд более простых. В нашей задаче предлагается схема вычислений, представленная на рис. 1.

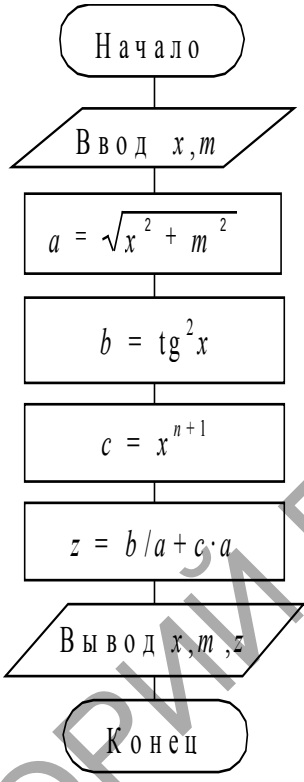


Рис. 1. Схема линейного процесса

Она содержит ввод и вывод исходных данных, линейный вычислительный процесс, вывод полученного результата. Заметим, что выражение $\sqrt{x^2+m^2}$ вычисляется только один раз. Введя дополнительные переменные *a*, *b*, *c*, мы разбили сложное выражение на ряд более простых.

Пример циклического процесса

Вычислить значение функции $y = \sin x$, представленной в виде разложения в ряд, с заданной точностью, т.е. до тех пор, пока разность между соседними слагаемыми не станет меньше заданной точности:

$$y = \sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots.$$

Схема алгоритма, приведенная на рис. 2, реализует циклический процесс, в состав которого (в блоке проверки $|E| < eps$) входит участок разветвления.

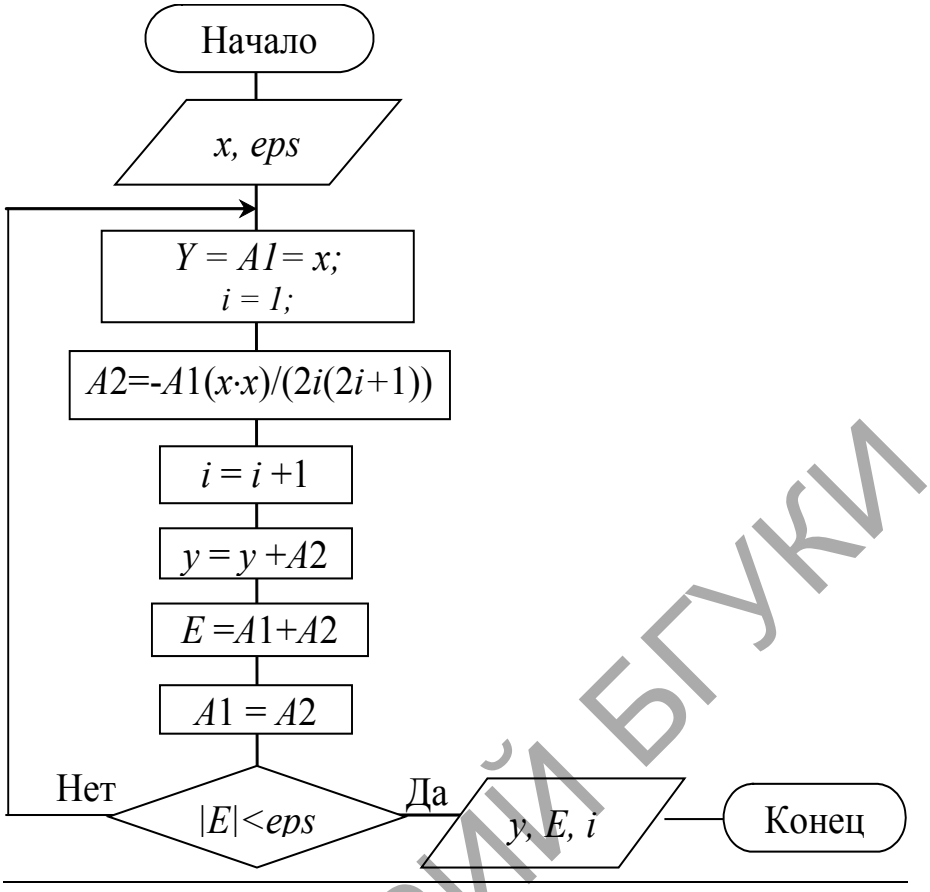


Рис. 2. Схема циклического алгоритма

Понятия алгоритма и программы не имеют четкого разграничения. Так, программа, записанная на алгоритмическом языке – это окончательный вариант алгоритма решения задачи, ориентированный на конкретного исполнителя (компьютер или язык программирования).

Тема 2. Основы технологии программирования

Программы и языки программирования. Программирование – процесс и искусство создания компьютерных программ с помощью языков программирования.

Язык программирования — формальная знаковая система для описания программы работы компьютера в форме, пригодной для трансляции и исполнения на компьютере. Язык программирования – это, прежде всего, инструмент деятельности, и на него в первую очередь оказывает влияние класс решаемых на нем задач. Хотя бы один язык нужно знать досконально, чтобы представлять многообразие имеющихся средств и иметь возможность сравнивать с другими.

Программа – описание на языке программирования структур данных и алгоритма решения задачи, автоматически переводимое, при помощи специальной программы-транслятора (компилятора или интерпретатора), на язык команд компьютера для последующего выполнения. Можно сказать, что компьютерная программа – один из способов реализации понятия алгоритма, а язык программирования – средство описания алгоритмов. *Компьютерная программа,* в отличие от абстрактного алгоритма, имеет *данные* – собственные элементы, над которыми она совершает действия, и которые являются ее составной частью. Алгоритмическая компонента программы – описание последовательности выполняемых действий – обычно состоит из *операторов*, задающих эту последовательность действий. Программа базируется на *наборе операций* над данными (арифметические операции, присваивание, проверка значения переменной и т.п.), соответствующем системе команд процессора, на котором она выполняется.

Структура данных – вид представления данных в программе, описание точки зрения пользователя на представление данных. Выбор подходящего представления данных – один из основных вопросов при проектировании программы. При этом под представлением данных понимается их описание на языке программирования в виде констант и переменных разной структуры. При решении задачи на компьютере, анализе исходных данных программы и ее результата, необходимо выбирать экономичный алгоритм решения, который и определит представление исходных, промежуточных и конечных данных. Неправильное представление данных может сделать программу ненадежной, неэкономичной, сложной и даже вообще неадекватной задаче.

Язык программирования содержит в себе компоненты, предназначенные для описания соответствующих частей программы:

•*средства описания данных*, позволяющие программисту определять различные формы представления данных (типы данных) и переменные разных типов;

•*набор операций* над основными типами данных (включая ввод-вывод), а также средства записи выражений;

•*набор операторов*, определяющих различные варианты порядка выполнения выражений в программе (последовательность, условие, повторение, блок);

•*средства разбиения программы* на независимые части – модули (функции), взаимодействующие между собой через программные интерфейсы.

Взаимосвязь алгоритма и данных в программе не является простой и линейной. Процесс выполнения любой программы можно рассматривать с

двух точек зрения: как последовательность выполнения команд, в которых содержится информация об операндах (данных), которые они обрабатывают – *поток команд* (*поток управления*). С другой стороны – любой элемент данных можно рассматривать как результат выполнения действий над исходными данными и как источник данных (операнд) для последующих результатов, т.е. в программе также присутствует логическая последовательность вычислений (преобразований данных), называемая *потоком данных.* Исторически сложилось так, что в традиционной (фон Неймановской) архитектуре в программе в явном виде задается последовательность команд, т.е. программа выглядит как *поток управления,* в котором алгоритмическая компонента является первичной (ведущей), а данные – вторичной (ведомой).

Этапы выполнения программ на ЭВМ. Технология подготовки и решения задачи на компьютере включает в себя не только «изготовление» программы, но и доказательство ее правильности:

•*ознакомление с поставленной задачей*– анализируются исходные данные, условия и цели решения задачи, формулируются требования к программе, разрабатывается точное описание того, что она должна делать и каких результатов необходимо достичь с ее помощью;

•*составление плана решения*– определение структуры данных, состава и последовательности требуемых преобразований, выбор способа решения, разработка алгоритма;

•*осуществление решения* – переработка согласно составленному плану входной информации в выходную, разработка программы с использованием той или иной технологии программирования;

•*проверка правильности решения задачи*– отладка, тестирование, счет, интерпретация результатов, получение выводов. Полученный вариант программы подвергается систематическому тестированию. Нельзя делать вывод, что программа работает правильно, лишь на том основании, что она выдала результаты (результаты не обязательно будут правильными, в программе может оставаться большое количество логических ошибок). Для каждой программы обязательно проводятся работы по обеспечению ее качества и эффективности, анализируются и улучшаются временные характеристики.

Существует 3 этапа прохождения программ на ЭВМ: исходный текст программы должен быть сохранен в файле. Этот файл подвергают обработке компилятором, и результатом является объектный код, автоматически сохраняемый в файле, программа обрабатывается компоновщиком – получается загрузочный модуль, автоматически сохраняемый в файле. Далее

программа идет на выполнение (в процессе могут быть подключены исходные данные), далее получаем результаты.

- компиляция – создание объектного файла *.obj, *.tpu
- компоновка – создает исполняемый файл *.exe
- выполнение – результат формируемые этапами

Исходный код программы. Объектный код программы. Способы трансляции. Компиляторы и интерпретаторы. Исходный код – текст компьютерной программы на каком-либо языке программирования или языке разметки, который может быть прочтен человеком. Исходный код транслируется в исполняемый код целиком до запуска программы при помощи компилятора, или может исполняться сразу при помощи интерпретатора. Исходный код либо используется для получения объектного кода, либо выполняется интерпретатором. Изменения никогда не выполняются над объектным кодом, только над исходным, с последующим повторным преобразованием в объектный. Другое важное назначение исходного кода – в качестве описания программы. По тексту программы можно восстановить логику ее поведения. Для облегчения понимания исходного кода используются комментарии.

Объектный код – некий текст программы, написанный без специфических для какого-либо языка программирования команд, но в котором оставлены названия функций и операций, названия переменных и другое. Объектный код, в отличие от исходного кода, позволяет формально подключится к нему из любого языка программирования. Сборка (компиляция) сложных программ происходит в два этапа - сначала идет подготовительный этап (написание объектного кода), а затем уже идет компиляция, чтобы получить исполняемый файл программы. У объектного кода есть еще одно преимущество - например, вы имеете сто файлов, которые подлежат компиляции. При запуске компиляции вы вдруг замечаете, что в одном из файлов происходит ошибка. В этом случае вы можете исправить только этот файл, пишите только для него объектный код заново и затем пересобираете программу, не нужно компилировать все сто файлов заново.

Машинный код – система команд конкретной вычислительной машины, которая интерпретируется непосредственно процессором или микропрограммами этой вычислительной машины. Каждая инструкция выполняет определенное действие, такое как операция с данными или переход к другому участку кода. Каждая исполнимая программа состоит из последовательности таких инструкций. Машинный код можно рассматривать как примитивный язык программирования или как самый низкий уровень представления скомпилированных или ассемблированных компьютерных

программ. Хотя вполне возможно создавать программы прямо в машинном коде, сейчас это делается редко в силу громоздкости кода и трудоемкости управления ресурсами процессора, за исключением ситуаций, когда требуется экстремальная оптимизация.

Для получения работающей программы текст необходимо перевести в машинный код для этого обращаются к программе переводчику, который называется *транслятором*.

Компилятор – получает объектный код. Это программы, которые обрабатывают весь программный текст, т е исходный код. Сначала он просматривает текст в поиске синтаксических ошибок, затем выполняется некоторый смысловой анализ, после чего текст автоматически переводится или транслируется на машинный язык. Не редко при генерировании машинного кода выполняется оптимизация с помощью набора методов, позволяющих повысить быстродействие программы. В результате законченная программа, которая называется объектным кодом, получается компактной и эффективной и может быть перенесена на другие компьютеры с процессором, поддерживающим соответствующий машинный код. *Интерприт*. Сразу берут операторы с текста программы, анализируют его структуру и затем сразу его выполняют. Только после успешного выполнения текущего оператора интерпретатор перейдет к следующему. При этом если один и тот же оператор повторяется многократно, интерпретатор всякий раз анализирует его как в первый. С помощью интерпритатора можно в любой момент остановить работу программы, исследовать содержимое памяти, организовать диалог с пользователем, т е интерпритатор полезен как инструмент на изучение программирования. В реальных системах программирования совмещены технологии компиляции и интерпритации.

Кроссплатформенность программных средств. Кроссплатформенное ПО – программное обеспечение, работающее более чем на одной аппаратной платформе или операционной системе. Если программа не предназначена для запуска на определенной платформе, но для этой платформы существует эмулятор платформы, базовой для данной программы, то программа может быть исполнена в среде эмулятора. Обычно исполнение программы в среде эмулятора приводит к снижению производительности по сравнению с аналогичными программами, для которых платформа является базовой, так как значительная часть ресурсов системы расходуется на выполнение функций эмулятора.

Понятие системы программирования. Интегрированная инструментальная среда для разработки программ. Системы программирования – это комплекс инструментальных программных средств,

предназначенный для работы с программами на одном из языков программирования. Системы программирования предоставляют сервисные возможности программистам для разработки их собственных компьютерных программ.

Системы программирования, как правило, включают в себя

– текстовый редактор (Edit), осуществляющий функции записи и редактирования исходного текста программы;

– загрузчик программ (Load), позволяющий выбрать из директория нужный текстовый файл программы;

– запускатель программ (Run), осуществляющий процесс выполнения программы;

– компилятор (Compile), предназначенный для компиляции или интерпретации исходного текста программы в машинный код с диагностикой синтаксических и семантических (логических) ошибок;

– отладчик (Debug), выполняющий сервисные функции по отладке и тестированию программы;

– диспетчер файлов (File), предоставляющий возможность выполнять операции с файлами: сохранение, поиск, уничтожение и т.п.

Важнейшим элементом современного программирования является программные средства, позволяющие создавать программы. Такой класс инструментального программного обеспечения получил *название интегрированных сред разработки* (*integrated developer environment – IDE*). Современные среды разработки программного обеспечения предоставляют разработчику-программисту высокоразвитые инструменты, которые существенным образом сокращают время проектирования и реализации программ, а также позволяют сделать разрабатываемое программное обеспечение значительно более качественным. Наряду с этим, использование интегрированной среды разработки существенным образом облегчает труд программиста.

Под интегрированной средой разработки понимают комплекс взаимосвязанных программных средств, который реализует развитый программный интерфейс между *компилятором* (*compiler*), *компоновщиком* (*linker*) и другими служебными программами, участвующими в процессе формирования бинарного кода приложения, с одной стороны, и разработчиком программ, с другой. Этот программный интерфейс реализуется в виде следующих элементов среды: редактор кода, редактор оконного интерфейса (конструктор форм), менеджер проектов, инспектор объектов программы, палитра визуальных компонентов и др.

Среда разработки позволяет управлять не только процессом кодирования приложения, но и берет на себя функции управления процессом

компоновки и отладки приложения. Так, все современные среды разработки имеют встроенные средства пошаговой отладки, расстановки контрольных точек в программе и точек останова.

В настоящее время существует большое число высокоразвитых сред разработки, ориентированных на компонентное объектно-ориентированное программирование. Различные среды разработки программного обеспечения ориентированы на различные сферы применения, используют в качестве языкового инструмента различные алгоритмические языки программирования. Однако, несмотря на функциональные различия, все современные среды разработки программ реализуют схожие концепции и подходы к разработке программного обеспечения и имеют в своем составе схожие инструментальные средства.

Хорошо известно, что на сегодняшний день законодателем «мод» в области разработки системного и прикладного программного обеспечения в целом и интегрированных сред разработки в частности является корпорация Microsoft. Наиболее известной и используемой интегрированной средой разработки этого производителя является пакет VisualStudio.NET, в котором поддерживаются такие языки программирования как VisualBasic.NET, VisualC++, VisualC# и VisualJ#.

На начальном уровне изучения современных концепций высокоуровневого программирования оптимальным выбором является среда разработки Visual Basic, которая использует в качестве базового одноименный язык программирования и предоставляет многочисленные компоненты, которые используются как строительные блоки приложения. Широкая известность и популярность этого языка программирования и соответствующей среды разработки обусловлена, в первую очередь, тем, что на протяжении более чем 20 лет этот инструмент программирования поддерживается на высоком уровне и обновляется корпорацией Microsoft.

Тема 3. Основы организации работы с данными при создании программ

Информационные системы создаются для достижения различных целей. Одной из главных целей является эффективная переработка данных в информацию или знания. Определим эти понятия.

Данные представляют собой элементарные описания предметов, событий, действий и транзакций, которые запомнены, классифицированы и сохранены, но не организованы для передачи какого – либо специального смысла. Элементы данных могут быть числовыми, алфавитно-числовыми, цифровыми, звуковыми или образными. База данных содержит хранящиеся элементы данных, организованные для доступа.

Информация – это данные, которые организованы так, что они имеют значение и ценность для получателя. Получатель (пользователь) интегрирует значения и выводит заключения и смыслы.

Знания состоят из данных или информации, которые организованы и обработаны с целью передачи понимания, накопленного опыта, результатов обучения и экспертизы таким образом, что они могут использоваться для решения текущих проблем или выполнения действий. Данные, которые обработаны для извлечения смыслов и для отражения прошлого опыта и экспертизы, обеспечивают пользователя организованным знанием, которое имеет очень высокую потенциальную ценность.

Эти три термина, особенно данные и информация, часто используются взаимозаменяемо. Данные, информация и знания могут быть для информационной системы входными или выходными.

Данные. Источники данных.

Данные в информационной системе поддержки решений могут включать документы, иллюстрации, карты, звуки и анимацию. Эти данные могут быть сохранены и организованы различными путями до и после их использования. Они также включают понятия, предметы и мнения (оценки). Данные могут быть предварительные, необработанные или обобщенные. Многие прикладные системы поддержки решений используют обобщенные или извлеченные данные, которые получают из трех основных источников: внутренних, внешних и персональных.

Внутренние данные хранятся в одном или более местах в корпорации. Это данные о людях, продукции, услугах и процессах. Информационная управляющая система может использовать как необработанные, так и обработанные данные (такие, как отчеты и сводки). Внутренние данные доступны через компьютерные сети организации.

Существует много источников *внешних данных*. Например, коммерческие базы данных, Интернет, спутниковая информация, фильмы, музыка, звуковая информация, иллюстрации, диаграммы, атласы, телевидение.

Постановления, нормативные акты и отчеты правительства являются главными источниками внешних данных.

Информация торгово-промышленных палат, локальных банков, исследовательских институтов, финансово – аналитических структур, биржевых сводок и другая, подобно наводнению обрушивается на пользователя информационной системы, вызывая у него информационные перегрузки.

Большинство внешних данных являются не относящимися к деятельности конкретной информационной системы. Поэтому

осуществляется целенаправленный мониторинг данных с целью извлечения необходимой информации и минимизации возможности пропуска и недооценки важности информации.

Пользователи информационных систем или другие сотрудники корпорации или предприятия могут использовать свои собственные экспертные знания и информацию для создания *персональных данных*. Они включают субъективные оценки продаж, мнения о возможных действиях конкурентов, интерпретации рыночной или производственной информации, прогнозные оценки и т.д.

Необходимость выделения данных из многих внутренних и внешних источников усложняет задачу построения информационной системы поддержки решений.

Необработанные данные могут быть собраны вручную или при помощи инструментов и сенсоров.

Типичными методами сбора данных являются: изучение во времени (посредством наблюдения), обследования (с использованием анкетирования), наблюдение (например, используя видеокамеры) и информация от экспертов (например, с использованием интервью).

Общеизвестна необходимость в достоверных и точных данных для любой системы поддержки решений. Однако в реальной жизни пользователи сталкиваются со слабоструктурированными задачами в зашумленных предметных областях с высоким уровнем неопределенности.

Данные должны быть доступны системе или система должна включать подсистему извлечения данных.

Как отмечалось, внешние данные стекаются в организацию из многих источников. Некоторые данные поступают на постоянной основе посредством межмашинного обмена по каналам связи между организациями, другие – посредством Интернет, который делает возможным доступ ко многим тысячам баз данных во всем мире.

Развитие Web – систем привело к использованию Web – браузеров для доступа к жизненно – важной информации для сотрудников и покупателей.

Другие Web – системы включают исполнительные информационные системы, системы поддержки, развернутые посредством Web - браузеров и системы управления базами данных (СУБД), которые обеспечивают данными непосредственно в формате, представляемом web – браузером с передачей посредством Интернет или интранет.

Большая тройка продавцов реляционных СУБД – компании Informix, Oracle и Sybase переработали свои основные продукты с целью приспособления клиент – серверных и Интернет интранет приложений, которые включали бы нетрадиционные или мультимедийные типы данных.

Структура данных и системы управления базами данных.

Сложность большинства корпоративных БД иногда делает стандартные операционные системы (ОС) компьютеров неадекватными эффективному интерфейсу между пользователем и БД. СУБД созданы для дополнения стандартных ОС возможностями более полной интеграции данных, сложных структур файлов, быстрого поиска и обмена, лучшей защиты данных. СУБД – это часть программного обеспечения для пополнения информации в БД и модернизации, удаления, манипулирования, хранения и поиска информации. СУБД в сочетании с языком моделирования является типичным инструментом развития системы, который используется при разработке информационной системы поддержки решений.

Отношения между многими индивидуальными записями, хранящимися в БД, могут быть выражены несколькими логическими структурами.

СУБД для выполнения своих функций разрабатываются с использованием таких структур.

Тремя основными структурами являются *реляционная*, *иерархическая* и *сетевая*. Более новыми структурами являются *объектно-ориентированные БД* и *мультимедийные БД.*

Рассмотрим две последние структуры подробнее.

Информационные системы поддержки решений в таких сложных предметных областях как интегрированное производство, требуют возможности доступа к сложным данным, которые могут включать иллюстрации и сложные отношения.

Ни иерархическая, ни сетевая, ни даже реляционная архитектура не может эффективно справляться с такими БД. Даже когда для создания и доступа в реляционной БД используется SQL, решения могут быть неэффективными.

Названные три типа БД являются алфавитно-числовыми. Но иногда для достижения лучших результатов требуется графическое представление.

Объектно-ориентированное управление данными базируется на принципах объектно-ориентированного программирования. Системы с объектно-ориентированными БД объединяют характеристики объектно-ориентированных языков, таких как Smalltalk или C++ с механизмом хранения данных и доступа к ним. Объектно-ориентированная СУБД позволяет анализировать данные на концептуальном уровне, который делает упор на естественные отношения между объектами.

Абстракция используется для установления наследственных иерархий, а описание и представление в сжатой форме позволяет проектировщику БД хранить обычные и процедурные коды внутри одних и тех же объектов.

Объектно-ориентированная СУБД определяет данные как объекты и представляет данные в сжатой форме в соответствии с их подходящей структурой и поведением.

Система использует иерархию классов и подклассов объектов. Структура (в терминах отношений) и поведение (в терминах методов и процедур) содержаться внутри объекта.

Объектно-ориентированные СУБД особенно полезны в распределенных информационных системах поддержки решений для очень сложных приложений и предметных областей.

Мультимедийные СУБД управляют данными в различных форматах (в дополнение к стандартному тексту или числовым полям). Эти форматы включают следующие образы: цифровые фотографии и формы компьютерной графики, такие как карты и .pic файлы; гипертекстовые образы; видеоклипы; звук и виртуальную реальность (многомерные образы).

Хранилище данных.

Современным организациям присуще использование как старых централизованных систем, так и новых распределенных систем. Широкое разнообразие технологий обеспечено также большим числом продавцов программных продуктов. Сталкиваясь с таким технологическим и коммерческим окружением, менеджеры должны использовать новые понятия в управляющих информационных технологиях. Одним из таких понятий является *складирование данных* (или хранение данных).

Определение понятия «хранилище данных» начинается с физического разделения оперативного окружения, поддерживающего решения. В сердцевине многих компаний используется хранилище оперативных данных, обычно извлекаемых из неавтономных систем обработки транзакций в режиме онлайн (OLTP – *online transaction processing* – оперативная обработка транзакций) и базирующихся на головных компьютерах (фейм – фрейм; mainframe).

OLTP – системы, например, для финансов, инвентаризации запасов или управления, также производят оперативные данные. В оперативном окружении доступ к данным, прикладные логические задачи и логика представления данных тесно взаимодействуют вместе, обычно в нереляционных БД. Эти нереляционные хранилища данных не очень способствуют эффективному поиску данных при поддержке решений.

Целью хранилища данных является установление такого ***репозитария данных***, который делает оперативные данные доступными в форме, которая приемлема для приложений в информационных системах поддержки решений. Как часть этого нового уровня доступности, процесс должен преобразовать детализированные по уровням оперативные данные в

реляционную форму, которая делает их более подходящими для обработки при поддержки решений.

Хранение данных (или хранение информации) – это понятие, предложенное и разработанное для обеспечения решения проблемы эффективного доступа к данным, описанным выше. Хранилище данных объединяет различные источники данных в простые источники для доступа конечного пользователя.

Существует несколько базовых структур для хранения данных. Основными являются двухрядные и трехрядные структуры. Вариант трехрядной архитектуры представлен на рисунке 3.

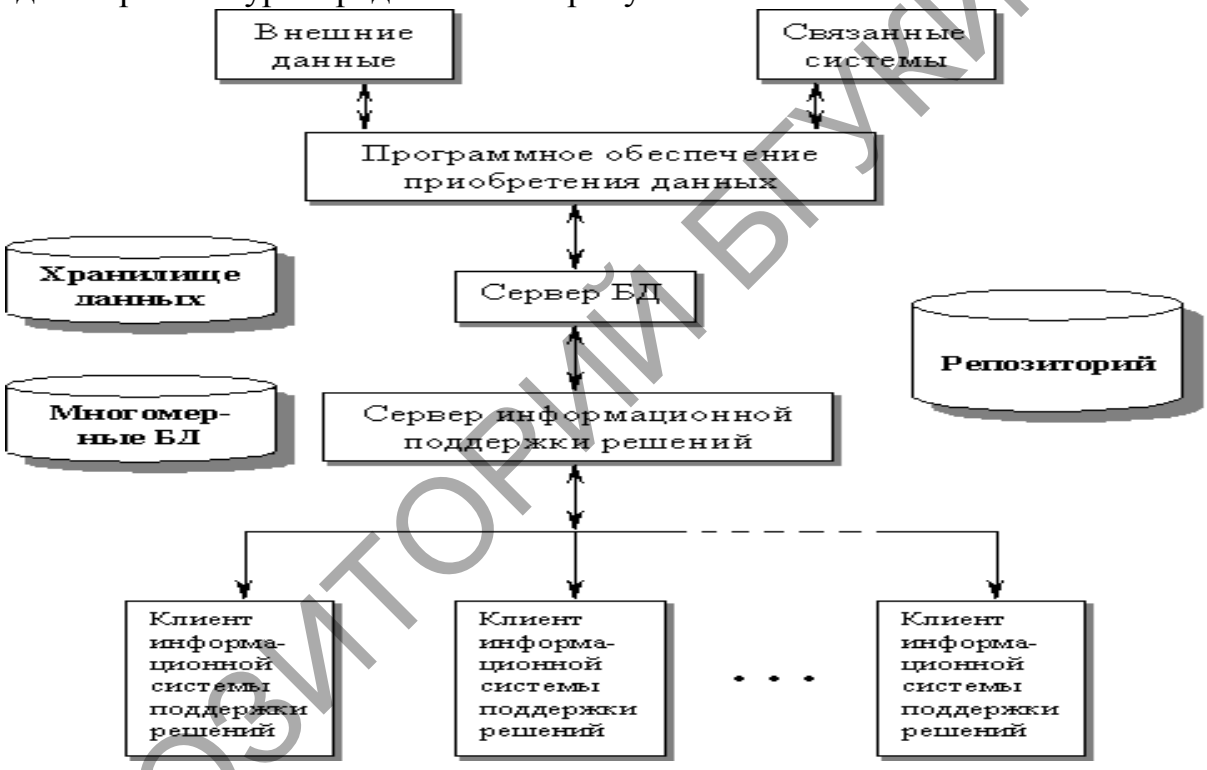


Рис.3. Трехрядная архитектура хранилища данных.

Перед размещением в хранилище данные, поступающие из внутренних (связанных) и внешних источников извлекаются, очищаются, фильтруются и суммируются посредством специального ПО. Далее данные снова обрабатываются и помещаются в дополнительную специальную многомерную БД (третий ряд в архитектуре), организованную для легкого многомерного представления. Пользователи информационной системы поддержки решений могут запрашивать сервер и осуществлять анализ.

В двухрядной архитектуре отсутствует многомерная БД или сервер.

Подобное хранение данных наиболее подходит для организаций, где:

– данные хранятся в различных системах;

– используется информационно-аналитический подход к менеджменту;

– имеется большая и разнообразная покупательская и клиентская база;

– одни и те же данные представлены по – разному в различных системах;

– данные хранятся в высокотехнических, трудных для расшифровки форматах.

OLAP: оперативная аналитическая обработка данных.

В течение многих лет информационные технологии концентрировались на построении систем поддержки обработки корпоративных транзакций. Такие системы должны быть визуально отказоустойчивыми и обеспечивать быстрый отклик. Эффективное решение было обеспечено OLTP, которые сосредотачивались на распределенном реляционном окружении БД.

Более поздним достижением в этой области явилось добавление архитектуры клиент – сервер. Было издано много инструментов для развития OLTP приложений.

Доступ к данным часто требуется как OLTP приложениям, так и информационным системам поддержки решений. К сожалению, попытка обслужить оба типа запросов может быть проблематична. Поэтому некоторые компании избрали путь разделения БД на OLTP тип и OLAP тип.

OLAP (Online Analytical Processing – оперативная аналитическая обработка) – это информационный процесс, который дает возможность пользователю запрашивать систему, проводить анализ и т.д. в оперативном режиме (онлайн). Результаты генерируются в течении секунд.

С другой стороны, в OLTP системе огромные объемы данных обрабатываются так скоро, как они поступают на вход.

OLAP системы выполнены для конечных пользователей, в то время как OLTP системы делаются для профессиональных пользователей ИС. В OLAP предусмотрены такие действия, как генерация запросов, запросы нерегламентированных отчетов, проведение статистического анализа и построение мультимедийных приложений.

Для обеспечения OLAP необходимо работать с хранилищем данных (или многомерным хранилищем), а также с набором инструментальных средств, обычно с многомерными способностями. Этими средствами могут быть инструментарий запросов, электронные таблицы, средства добычи данных (Data Mining), средства визуализации данных и др.

В основе концепции OLAP лежит принцип многомерного представления данных. Э. Кодд рассмотрел недостатки реляционной модели, в первую очередь, указав на невозможность объединять, просматривать и анализировать данные с точки зрения множественности измерений, то есть самым понятным для корпоративных аналитиков способом, и определил общие требования к системам OLAP, расширяющим функциональность

реляционных СУБД и включающим многомерный анализ как одну из своих характеристик.

В большом числе публикаций аббревиатурой OLAP обозначается не только многомерный взгляд на данные, но и хранение самих данных в многомерной БД. Вообще говоря, это неверно, поскольку сам Кодд отмечает, что реляционные БД были, есть и будут наиболее подходящей технологией для хранения корпоративных данных. Необходимость существует не в новой технологии БД, а скорее, в средствах анализа, дополняющих функции существующих СУБД и достаточно гибких, чтобы предусмотреть и автоматизировать разные виды интеллектуального анализа, присущие OLAP.

По Кодду, многомерное концептуальное представление представляет собой множественную перспективу, состоящую из нескольких независимых измерений, вдоль которых могут быть проанализированы определенные совокупности данных. Одновременный анализ по нескольким измерениям определяется как многомерный анализ. Каждое измерение включает направления консолидации данных, состоящие из серии последовательных уровней обобщения, где каждый вышестоящий уровень соответствует большей степени агрегации данных по соответствующему измерению. Так измерение Исполнитель может определяться направлением консолидации, состоящим из уровней обобщения «предприятие – подразделение – отдел - служащий». Измерение Время может даже включать два направления консолидации – «год – квартал – месяц - день» и «неделя - день», поскольку счет времени по месяцам и по неделям несовместим. В этом случае становится возможным произвольный выбор желаемого уровня детализации информации по каждому из измерений. Операция спуска соответствует движению от высших ступеней консолидации к низшим; напротив, операция подъема означает движение от низших уровней к высшим.

Кодд определил 12 правил, которым должен удовлетворять программный продукт класса OLAP. Эти правила:

1. Многомерное концептуальное представление данных.
2. Прозрачность.
3. Доступность.
4. Устойчивая производительность.
5. Клиент – серверная архитектура.
6. Равноправие измерений.
7. Динамическая обработка разреженных матриц.
8. Поддержка многопользовательского режима.
9. Неограниченная поддержка кроссмерных операций.
10. Интуитивное манипулирование данными.
11. Гибкий механизм генерации отчетов.

12. Неограниченное количество измерений и уровней агрегации.

Набор этих требований, послуживший фактическим определением OLAP, следует рассматривать как рекомендательный, а конкретные продукт оценивать по степени приближения к идеально полному соответствию всем требованиям.

Интеллектуальный анализ данных.

Интеллектуальный анализ данных (ИАД), или Data Mining, - термин, используемый для описания открытия знаний в базах данных, выделения знаний, изыскания данных, исследования данных, обработки образцов данных, очистки и сбора данных; здесь же подразумевается сопутствующее ПО. Все эти действия осуществляются автоматически и позволяют получать быстрые результаты даже непрограммистам.

Запрос производится конечным пользователем, возможно на естественном языке. Запрос преобразуется в SQL – формат. SQL запрос по сети поступает в СУБД, которая управляет БД или хранилищем данных. СУБД находит ответ на запрос и доставляет его назад. Пользователь может затем разрабатывать презентацию или отчет в соответствии со своими требованиями.

Многие важные решения в почти любой области бизнеса и социально сферы основываются на анализе больших и сложных БД. ИАД может быть очень полезным в этих случаях.

Методы интеллектуального анализа данных тесно связаны с технологиями OLAP и технологиями построения хранилищ данных. Поэтому наилучшим вариантом является комплексный подход к их внедрению.

Для того чтобы существующие хранилища данных способствовали принятию управленческих решений, информация должна быть представлена аналитику в нужной форме, то есть он должен иметь развитые инструменты доступа к данным хранилища и их обработки.

Очень часто информационно – аналитические системы, создаваемые в расчете на непосредственное использование лицами, принимающими решения, оказываются чрезвычайно просты в применении, но жестко ограничены в функциональности. Такие статические системы называются Информационными системами руководителя. Они содержат в себе предопределенные множества запросов и, будучи достаточными для повседневного обзора, неспособны ответить на все вопросы к имеющимся данным, которые могут возникнуть при принятии решений. Результатов работы такой системы, как правило, являются многостраничные отчеты, после тщательного изучения которых у аналитика появляется новая серия вопросов. Однако каждый новый запрос, непредусмотренный при проектировании такой системы, должен быть сначала формально описан,

закодирован программистом и только затем выполнен. Время ожидания в таком случае может составлять часы и дни, что не всегда приемлемо. Таким образом, внешняя простота статистических ИС поддержки решений, за которую активно борется большинство заказчиков информационно – аналитических систем, оборачивается потерей гибкости.

Динамические ИС поддержки решений, напротив, ориентированы на обработку нерегламентированных (ad hoc) запросов аналитиков к данным. Работа аналитиков с этими системами заключается в интерактивной последовательности формирования запросов и изучения их результатов.

Но динамические ИС поддержки решений могут действовать не только в области оперативной аналитической обработки (OLAP). Поддержка принятия управленческих решений на основе накопленных данных может выполняться в трех базовых сферах.

1. Сфера детализированных данных. Это область действия большинства систем, нацеленных на поиск информации. В большинстве случаев реляционные СУБД отлично справляются с возникающими здесь задачами. Общепризнанным стандартом языка манипулирования реляционными данными является SQL. Информационно – поисковые системы, обеспечивающие интерфейс конечного пользователя в задачах поиска детализированной информации, могут использоваться в качестве надстроек как над отдельными базами данных транзакционных систем, так и над общим хранилищем данных.

2. Сфера агрегированных показателей. Комплексный взгляд на собранную в хранилище данных информацию, ее обобщение и агрегация и многомерный анализ являются задачами систем OLAP. Здесь можно или ориентироваться на специальные многомерные СУБД, или оставаться в рамках реляционных технологий. Во втором случае заранее агрегированные данные могут собираться в БД звездообразного вида, либо агрегация информации может производиться в процессе сканирования детализированных таблиц реляционной БД.

3. Сфера закономерностей. Интеллектуальная обработка производится методами интеллектуального анализа данных, главными задачами которых являются поиск функциональных и логических закономерностей в накопленной информации, построение моделей и правил, которые объясняют найденные аномалии и/или прогнозируют развитие некоторых процессов.

Полная структура информационно – аналитической системы построенной на основе хранилища данных, показана на рисунке 4. В конкретных реализациях отдельные компоненты этой схемы часто отсутствуют.

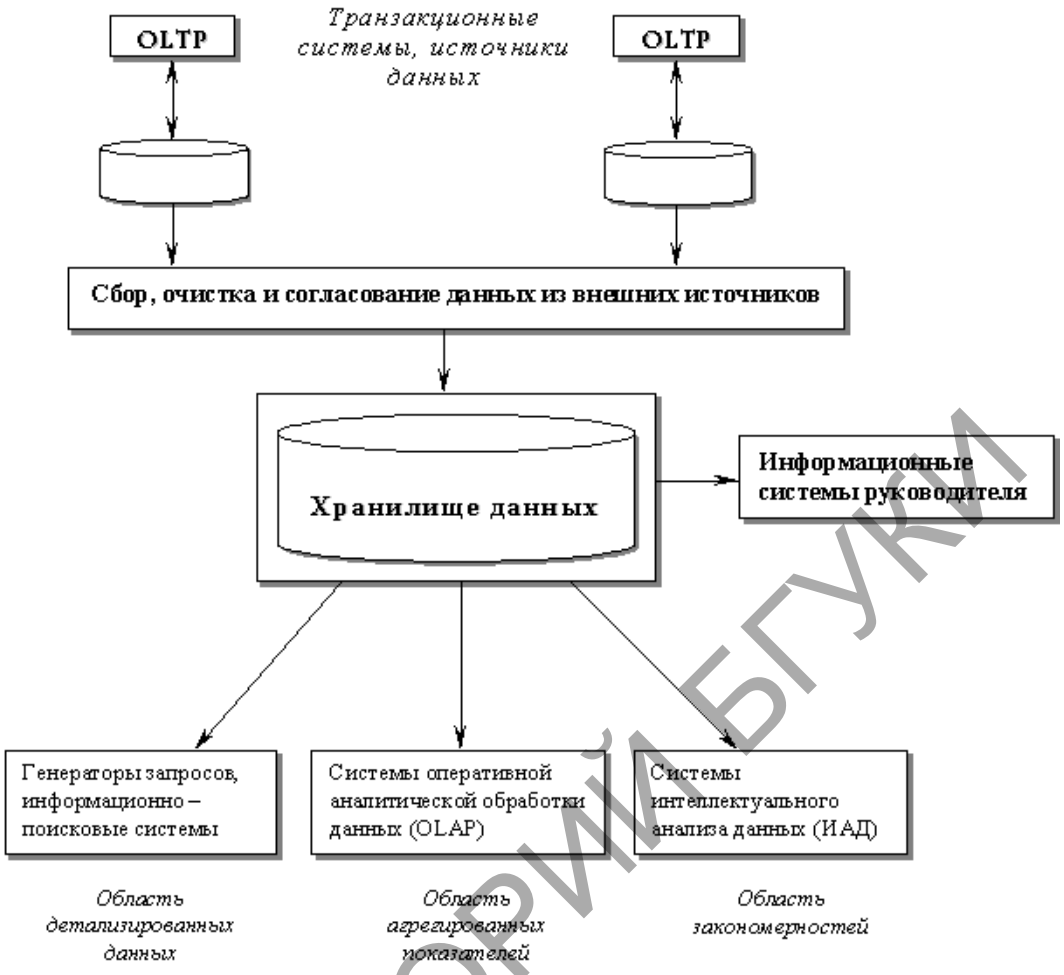


Рис. 4. Структура корпоративной информационно – аналитической системы.

Интеллектуальные базы данных.

Развитие приложений ИС требует реализации более легкого и удобного доступа к базам данных.

Технологии ИИ, особенно ЭС и искусственные нейронные сети (ИНС), могут сделать доступ и манипуляции в сложных БД проще. Одним из путей является усиление роли СУБД в обеспечении этого, совместно со способностью выведения заключений, что в результате получило общее название *интеллектуальная БД*.

Одним из вариантов интеграции ЭС и БД показан на рисунке 5.

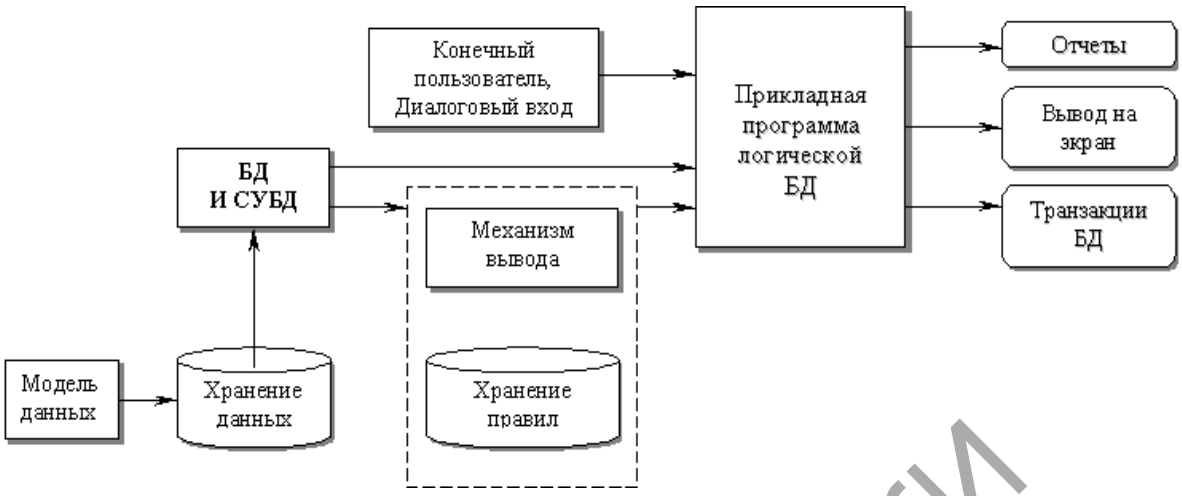


Рис. 5. Структура интеллектуальной базы данных, представляющая один из способов интеграции ЭС и БД

Трудности в соединении ЭС с большими БД являются главной проблемой даже для больших корпораций. Многие продавцы ПО, осознавая важность такой интеграции, развивают свою программную продукцию для ее поддержки. Примером такого продукта является реляционная СУБД компании Oracle, которая объединяет функциональность ЭС с БД и представляет в форме оптимизатора запросов, которые отбирает наиболее эффективные пути следования запросов БД.

Оптимизация важна для пользователей, т.к. с такой способностью им нужно знать только несколько правил и команд для использования БД.

Одним из главных текущих направлений в разработке коммерческих программ ИИ компании IBM является обеспечение подсистемы обработки знаний для работы с БД, которая дает возможность пользователям выделить информацию из БД и передать ее в базу правил ЭС в нескольких различных структурах представления знаний.

Другой продукт – это KEE Connection (Intelli Corporation), который переводит команды KEE (KEE – Knowledge Engineering Environment) в запросы БД и автоматически поддерживает тракт данных, флуктуирующих туда и обратно между базой знаний KEE и реляционной БД, использующей SQL. Другими преимуществами такой интеграции являются способности использовать символьное представление данных и улучшения в конструкции, операциях и поддержании СУБД.

Некоторые программные инструменты для добычи данных включают интеллектуальные системы, которые поддерживают интеллектуальный поиск. Интеллектуальная добыча и анализ данных (ИАД) позволяет открыть

информацию в хранилищах данных, когда запросы и отчеты не могут быть обнаружены.

Инструменты ИАД находят образцы в данных и выводят из них правила. Эти образцы и правила могут быть использованы для руководства при принятии решений и прогнозировании результатов этих решений. ИАД может ускорить анализ путем сосредоточения внимания на наиболее важных переменных.

Пять типов информации может быть применено при ИАД: ассоциации, последовательности, классификации, кластеры и прогнозирование.

Основными типами программных инструментариев, используемых в ИАД, являются:

– рассуждения на основе прецедентов;

– нейронные вычисления;

– интеллектуальные агенты;

– другие средства: деревья решений, ролевая индукция, визуализация данных.

Знания в искусственном интеллекте.

База знаний.

Знание – это теоретическое и практическое понимание предмета или области.

Знаниями принято также называть хранимую в компьютере информацию, формализованную в соответствии с определенными структурными правилами, которую компьютер может автономно использовать при решении проблем на основе логического вывода. В общепринятом смысле термин «знания» стал чрезвычайно популярным, однако этому термину сложно дать определение, поскольку он включает в себя большой частью философские элементы. Если придерживаться буквального смысла термина, то его общие концепции понимаются с трудом, но когда речь заходит о представлении знаний в компьютере, то имеется в виду упомянутый ограниченный смысл. То есть, обработка знаний в компьютере представляет собой обработку их содержимого правилами преобразования тех форм, которыми описываются знания в машине.

Главная характеристика, которая отличает ИИ от других информационных систем – это то, что основной акцент ИИ – это обработка знаний (в большей степени, чем обработка данных или информации). В настоящее время знания признаны главным ресурсом организации.

Несмотря на то, что компьютер не может иметь (пока) разнообразный опыт или обучаться, как человеческий разум, он может использовать знания, данные ему людьми (экспертами). Такие знания содержат факты, понятия, теории, эвристические методы, процедуры и отношения.

Знания – это также информация, которая организована и проанализирована с целью сделать ее понятной и применимой для решения задачи или принятия решений. Коллекция совместно организованных знаний, относящихся к задачам, решаемым в системе ИИ, называется базой знаний (БЗ).

Большинство БЗ ограничены в некоторой специальной, обычно узкой предметной области, в которой они сосредоточены. При создании БЗ технология ИИ позволяет встраивать в компьютер механизм и способности вывода, основывающиеся на фактах и отношениях, содержащихся в БЗ.

Система управления базой знаний

Методы представления знаний, развитые в базах данных и в искусственном интеллекте, не вполне удовлетворительны для использования при разработке промышленных систем управления базами знаний СУБЗ, которые призваны обеспечивать создание, сопровождение и применение баз знаний в интеллектуальных системах.

Для построения СУБЗ как одного из важнейших инструментальных средств новой технологии необходима интеграция методов представления знаний в БД и ИИ.

Опыт работ в области ИИ и в технологии баз данных позволяет сформулировать основные требования к такой интегрированной системе.

Поскольку СУБЗ представляет собой инструментальное средство, с ней работают в первую очередь программисты – разработчики интеллектуальных систем и администраторы баз знаний (инженеры знаний) – специалисты, отвечающие за проектирование и сопровождение баз знаний в актуальном состоянии, т.е. в таком состоянии, которое правильно отражает внешнюю среду.

Основная задача программистов – разработчиков – создание процедурной части интеллектуальной системы, работающей на основе БЗ. Для решения этой задачи имеются развитые инструментальные средства, не обеспечивающие, однако, необходимого уровня эффективности при работе с большими базами знаний. Интеграция должна преодолеть этот недостаток.

Цель интеграции для разработчиков интеллектуальных систем – обеспечить создание единых инструментальных (языковых) средств, успешно и эффективно реализующих методы доступа к информации и обработки ее, типичные и для искусственного интеллекта и для технологии баз данных, и не зависящие от того, где эта информация размещается. Иначе говоря, применяемые методы физической организации базы знаний (размещение ее в многоуровневой памяти компьютера) должны быть прозрачны для программиста – разработчика. В этом случае примитивы

доступа к информации (типичные для той или иной области) выбираются программистом только из соображений удобства.

Цель интеграции для администраторов БЗ – обеспечить ряд средств, представленных в основном в технологии баз данных, но приспособленных к требованиям СУБЗ.

Это, прежде всего:

– Средства автоматизации логического и физического проектирования БЗ, обеспечивающие, в частности, помощь в выборе способов хранения отдельных фрагментов БЗ в соответствии с критерием максимальной эффективности функционирования СУБЗ;

– Средства поддержания логической и физической целостности СУБЗ в процессе ее эксплуатации, т.е. обеспечение надежной работы системы в условиях возможных сбоев технических и программных средств компьютера, ошибок пользователей;

– Средства реорганизации БЗ для повышения эффективности работы СУБЗ и в связи с глобальными изменениями системы знаний. Все эти средства имеют прототипы в технологии баз данных.

– Опыт, накопленный в технологии баз данных, свидетельствует о необходимости таких средств и в технологии баз знаний.

Обработка знаний.

Обработка знаний в компьютере представляет собой обработку их содержимого правилами преобразования тех форм, которыми описываются знания в машине. Следовательно, при обработке знаний наиболее фундаментальной и важной проблемой является прежде всего описание смыслового содержимого проблем широкого диапазона, а также наличие такой формы описания знаний, которая гарантирует, что обработка их содержимого формальными правилами преобразования будет осуществляться правильно. Эта проблема называется проблемой представления знаний.

Обработка знаний является одной из областей обработки информации. Обсуждение технологии обработки информации в течение многих лет вплоть до настоящего времени ведется в основном вокруг вычислительных механизмов машины Тьюринга – фон Неймана, которая вплотную подошла к своей оптимальной форме. Экспертные и интеллектуальные системы, являющиеся практически те же методы, а их функции базируются на тех же механизмах.

Однако при анализе этих проблем необходимо проникнуть в суть вопроса с позиции более широкого исследования процесса решения проблемы. Это связано с тем, что желаемая структура систем обработки знаний, обладающих возможностями решения сложных проблем, еще не

определена в полной мере. Кроме того, при интеллектуальной обработке информации преследуется цель создания нового стиля обработки на основе когнитивных человеко-машинных отношений, совершенно отличных от тех, что приняты в машине Тьюринга – фон Неймана. Следовательно для утверждения нового стиля обработки информации необходимо сначала проанализировать поведение человека при решении проблем, а затем посмотреть, какую часть процесса решения можно автоматизировать средствами обработки знаний, насколько эффективно можно этими средствами поддерживать решение проблем и что будет представлять из себя система, которую удастся воплотить в реальную действительность?

Сегодня системы обработки знаний, достигшие в своем развитии коммерческого уровня, реализованы на компьютерах традиционного типа. Однако реальное воплощение в жизнь машинной обработки, оперирующей с представлениями информации, ориентированными на обработку знаний, возможно и другими, отличными от фон – неймановскими средствами. Смысл заключается в привнесении новых принципов в обработку информации.

Интенсивное количественное и качественное развитие информационных и компьютерных технологий, проявление новых когнитивных концепций, позволяет говорить о возможности практической реализации альтернативных методов и подходов к обработке знаний.

Отличительная черта обработки знаний состоит в возможности изменения человеко-машинных отношений и в соответствии с этим – становление нового стиля решения проблем.

Инженерия знаний.

Инженерия знаний представляет собой совокупность моделей, методов и технических приемов, нацеленных на создание систем, которые предназначены для решения проблем с использованием знаний. Знания – это информация с ограниченной семантикой, однако с позиции прикладных аспектов необходимо, чтобы знания имели такую форму, которой была бы в определенной степени свойственна свобода достижения поставленной цели. В какой именно степени допустима эта свобода, или каким условиям должны отвечать знания, включая и их описательные возможности, зависит от области их приложения. В сфере технического применения и в экономике используется самая разнообразная среда представления, и помимо языкового описания она включает рисунки, математические формулы и т.п.

Хотя языковое представление и ограничено сравнительно простыми формализмами, оно не всегда удобно для технической и экономической областей. Это связано с их специфическим характером, т.к. в них все определяется фактами и объективной реальностью.

В дальнейшем изложении языковое описание, требуемое в прикладных областях информации (включая язык в широком его понимании и графику), будет называться языком представления знаний. Для использования подобной информации в виде знаний требуются интеллектуальные функции, превосходящие пока возможности современных компьютеров. Представление знаний, их обработка и использование, рассматриваемое применительно к конкретной прикладной области, является предметом инженерии знаний.

Инженерия знаний заняла свое место как технология применения знаний, когда вышла из недр ИИ и продолжала интенсивно развиваться все последние года.

Существом ИИ можно считать научный анализ и автоматизацию интеллектуальных функций человека. Однако для большинства проблем общей реальностью является трудность их машинного воплощения.

Исследования по ИИ позволили утвердиться во мнении, что подлинно необходимыми для решения проблем являются знания экспертов. То есть, если создать систему, способную запоминать и использовать знания экспертов, то она найдет применение в практической деятельности.

И когда исследователи по ИИ действительно создали подобного ряда системы в конце 60-х и начале 70-х годов прошлого века, все эти воззрения были подтверждены.

Это системы DENDRAL, а позднее MYCIN, созданные под руководством Э. Фейгунбаума в Стэнфордском университете США, Поскольку эти системы накапливают в памяти компьютера знания экспертов и используют эти знания для решения проблем, извлекая их при необходимости из памяти, то они получили название экспертных, а профессор Э. Фейгенбаум, являющийся одним из создателей экспертных систем (ЭС), выдвинул для данной области техники название «инженерия знаний».

Фактически инженерия знаний – это методология ЭС, которая охватывает методы добычи, анализа и выражения в правилах знаний экспертов. Развитие ЭС создало инженерию знаний – процесс построения интеллектуальных систем.

Инженерия знаний тесно связана со всем процессом разработки интеллектуальных информационных систем в целом и ЭС в частности – от возникновения замысла до его реализации и совершенствования.

Главными элементами инженерии знаний являются использование операций типа обобщение, генерация гипотез для индуктивных выводов, подготовка новых программ самими компьютерными программами и т.д.

Слово engineering в английском означает искусная обработка предметов, изобретение или создание чего-либо. Следовательно, работу по оснащению программ специальными экспертными знаниями из проблемной области, выполняемую человеком, либо компьютером (программой), также можно назвать инженерией знаний.

Тема 4. Основы объектно-ориентированного программирования

Объектно-ориентированное программирование (ООП) – это методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является реализацией определенного ***класса*** (типа особого вида), причем классы образуют иерархию на принципах наследуемости. Под термином ***объект*** понимается «осязаемая сущность, которая четко проявляет свое поведение».

Объект характеризуется

- совокупностью всех своих свойств и их текущих значений и
- совокупностью допустимых для него действий (их называют *методами*)

В качестве примера объекта можно привести животное, свойствами которого могут быть голова (большая, маленькая, …), уши (длинные, короткие, … или другие), а методами (умение принимать пищу, стоять, сидеть, идти, бежать и т.п.).

Объектно-ориентированный стиль является наиболее приемлемым для широчайшего круга приложений. Его концептуальной базой является *объектная модель*. Она имеет четыре главных элемента (без любого из них модель не будет объектно-ориентированной):

- абстрагирование
- инкапсуляция
- модульность
- иерархия

И три дополнительных элемента:

- типизация
- параллелизм
- сохраняемость

Абстрагирование является одним из основных методов, используемых для решения сложных задач. Абстрагирование проявляется в нахождении сходств между определенными объектами, ситуациями или процессами реального мира, и в принятии решений на основе этих сходств, отвлекаясь на время от имеющихся различий. Хорошей является такая абстракция, которая подчеркивает детали, существенные для рассмотрения и использования, и

опускает те, которые на данный момент несущественны. Абстракция позволяет формализовать описание.

Пример:

В тепличном хозяйстве управление режимом работы парниковой установки — очень ответственное дело. В больших хозяйствах для решения этой задачи часто используют автоматические системы, которые контролируют и регулируют целый ряд факторов: температуру, влажность, освещение...

Одна из ключевых абстракций в такой задаче — датчик температуры. Датчик температуры — это объект, который способен измерять температуру там, где он расположен.

Что такое температура? Это числовой параметр, имеющий ограниченный диапазон значений и определенную точность, означающий число градусов по Фаренгейту, Цельсию или Кельвину.

Что такое местоположение датчика? Это некоторое идентифицируемое место в теплице, температуру в котором нам необходимо знать. Для датчика температуры существенно не столько само местоположение, сколько тот факт, что данный датчик расположен именно в данном месте и это отличает его от других датчиков.

Или клумба с различными цветами:

Общее – цветок, состоящий из стебля, бутона (лепестки, тычинки, пестики, листья и т.д.).

Или же еще строения. Общее понятие – здание, а далее идут признаки, т.е. наличие определенных свойств: этажность, ширина, длина, высота. Здесь не важен цвет, материал.

Абстракция "помогает людям думать о том, что они делают", инкапсуляция "позволяет легко перестраивать программы".

Инкапсуляция - это процесс отделения друг от друга элементов объекта, определяющих его устройство и поведение.

Пример:

Чтобы понять на верхнем уровне действие фотосинтеза, вполне допустимо игнорировать такие подробности, как функции корней растения или химию клеточных стенок.

При проектировании базы данных принято писать программы так, чтобы они не зависели от физического представления данных; вместо этого сосредотачиваются на схеме, отражающей логическое строение данных.

В обоих случаях объекты защищены от деталей реализации объектов более низкого уровня.

Таким образом, суть инкапсуляции: спрятать детали представления максимально вглубь, чтобы любые их изменения не влияли на основные свойства.

Необходимо преобразование данных для получения результата; обмен данными проходит на уровне внешних свойств.

Модульность

Разделение программы на модули позволяет уменьшить ее сложность за счет независимой разработки и тестирования. Структура модуля должна быть достаточно простой для восприятия; реализация каждого модуля не должна зависеть от реализации других модулей; должны быть приняты меры для облегчения процесса внесения изменений там, где они наиболее вероятны.

Интерфейс + тело = реализация

В интерфейсе собраны все свойства, которые необходимо реализовать: сложение, вычитание, умножение, деление.

В интерфейсе должно быть имя числа, над которым можно выполнять названные операции.

Плавающее число – число с плавающей точкой. Характеризуется диапазоном и точностью представления.

Символьные строки – набор символов, которые могут использоваться для ее представления.

Сложение строк – сцепление, присоединение. Совсем другая семантика. Умножение, деление смысла не имеет.

Следовательно, выявление сущности и инкапсуляция приводят к группированию наиболее общих свойств объектов. Важно какие строки и в каком виде могут представляться.

Абстракция — вещь полезная, но всегда, кроме самых простых ситуаций, число абстракций в системе намного превышает наши умственные возможности. Инкапсуляция позволяет в какой-то степени устранить это препятствие, убрав из поля зрения внутреннее содержание абстракций. Модульность также упрощает задачу, объединяя логически связанные абстракции в группы. Но этого оказывается недостаточно.

Значительное упрощение в понимании сложных задач достигается за счет образования из абстракций иерархической структуры.

Иерархия — это упорядочение абстракций, расположение их по уровням.

Основными видами иерархических структур применительно к сложным системам являются:

- структура классов (иерархия «is-a»);
- структура объектов (иерархия «part of»).

Структура классов

Наследование означает такое отношение между классами (отношение родитель/потомок), когда один класс заимствует структурную или функциональную часть одного (*одиночное наследование*) или нескольких других классов (*множественное наследование*). Общая часть структуры и поведения сосредоточена в наиболее общем суперклассе. По этой причине о наследовании говорят как об иерархии *обобщение-специализация*. Суперклассы отражают наиболее общие, а подклассы - более специализированные абстракции, в которых члены суперкласса могут быть дополнены, модифицированы и даже скрыты. Принцип наследования позволяет упростить выражение абстракций, делает проект менее громоздким и более выразительным.

Семантически, наследование описывает отношение типа «is-a».

Пример:

медведь есть млекопитающее;
дом есть недвижимость;
«быстрая сортировка» есть сортирующий алгоритм.

Таким образом, наследование порождает иерархию «обобщение-специализация», в которой подкласс представляет собой специализированный частный случай своего суперкласса.

Пример:

Рассмотрим различные виды растений, выращиваемых в огородной системе. Есть общий план выращивания растений. Однако разные культуры требуют разных планов. При этом планы для фруктов похожи друг на друга, но отличаются от планов для овощей или цветов. Имеет смысл ввести на новом уровне абстракции обобщенный «фруктовый» план, как наследника общего плана выращивания, включающий указания по опылению и сборке урожая.

Структура объектов

Отношение "partof" вводит иерархию агрегации. Агрегация позволяет физически сгруппировать логически связанные структуры. В иерархии "partof" класс находится на более высоком уровне абстракции, чем любой из использовавшихся при его реализации. В отличие от иерархии классов, где вышележащая абстракция является обобщением, а нижестоящая – специализацией.

Пример:

иерархия "part of" Класс *Garden* стоит на более высоком уровне, чем класс *Plant*.

иерархия "is-a" Класс *Flower* находится на более высоком уровне абстракции, чем класс *Plant*.

Пример: Роза, за признаки примем шипы на стебле. Т.е. роза есть цветок + шипы. Можно построить такую сущность как красная роза и желтая роза (отличие в цвете). Здесь же можно выделить другие абстракции.

Так роза еще может быть дикой, т.е. дикая роза (шиповник), которую будем определять как розу с плодами.

Таким образом, каждая следующая сущность определяется дополнением некоторых других свойств.

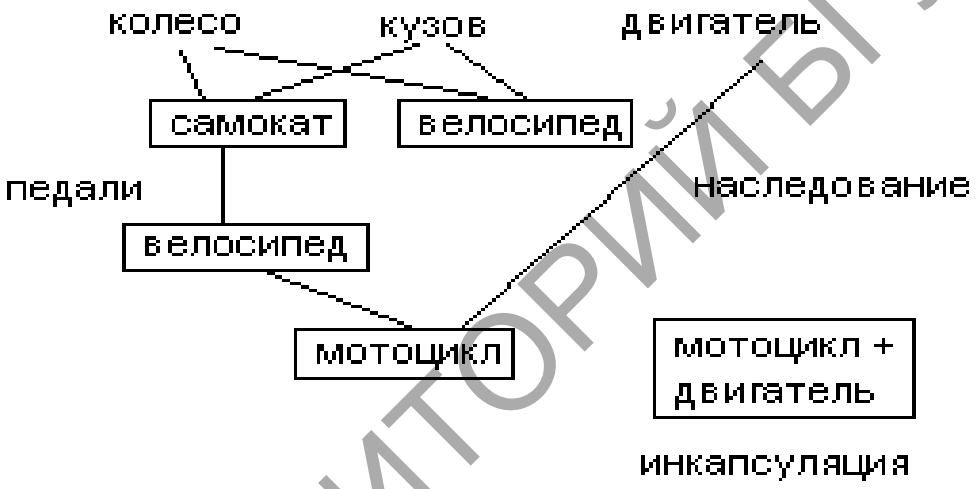
Переход от общего к конкретному.

Наземное транспортное средство (ТС)

Грузовое ТС Пассажирское ТС

Универсальное ТС

Пример:



Если выбирать между простым наследованием и инкапсуляцией, то лучше использовать простое наследование, а если между множественным наследованием (где возникают сложности в отладке) и инкапсуляцией, то – инкапсуляцию.

Объектно-ориентированная методология программирования базируется на следующих принципах:

- инкапсуляция;
- наследование;
- полиморфизм.

Инкапсуляцией называется объединение в одном объекте как свойств, так и методов (действий над ними)**.**

Наследование предполагает существование иерархии классов (объектных типов), в которой между классами, расположенными на разных уровнях иерархии, имеют место отношения наследования свойств и методов

- от объектов вышележащих (*родительских*) к объектам нижележащих (*дочерних*) классов.

Наследование – это такое отношение между объектами, когда один объект повторяет структуру и поведение другого, принадлежащего к более высокому уровню в иерархии.

Наследование можно определить так:

Родительские классы называют просто родителями (*прародителями*), а дочерние –*потомками*.

Классы верхних уровней иерархии, как правило, не имеют конкретных экземпляров объектов. Не существует, например, конкретного живого организма (объекта), который бы сам по себе назывался «Млекопитающее» или «Насекомое». Такие классы называются *абстрактными*. Конкретные экземпляры объектов, или просто объекты, принадлежат нижележащим уровням иерархии.

Полиморфизм – это способность объектов разных классов в иерархии выполнять одно и то же действие по-своему. (Или иначе, полиморфизм – это придание одного и того же имени методам для объектов разных классов в иерархии, хотя действия в методах различаются). При объектно-ориентированном программировании программист только указывает, какому объекту, какое из присущих ему действий, требуется выполнить, и, однажды объявленные (описанные в программе), объекты сами будут выполнять их характерными именно для них способами.

Например, действие «бежать» свойственно большинству животных, однако каждое из них (лев, слон, черепаха) выполняет это действие различным образом. В данном примере действие «бежать» будет называться полиморфическим действием, а многообразие форм проявления этого действия – полиморфизмом.

Тема 5. Основы Веб-программирования

Большинство Web-страниц представляют собой гипертекстовые документы, отформатированные с помощью специального языка разметки документов HTML (Hyper Text Markup Language). Под гипертекстом понимают способ представления информации при помощи связей между документами, а также внутри одного документа. В Web-документе связь представляет собой URL-ссылку (унифицированный указатель ресурсов), указывающую на другую Web-страницу или любой другой информационный ресурс сети.

Язык разметки гипертекста состоит из набора элементов, которые описывают структуру документа и управляют его отображением. Разметка

документов производится с помощью специальных флагов разметки, называемых тегами (tag – метка), которые интерпретируются Web-браузерами при загрузке HTML-документа. Теги являются инструкциями для браузера и определяют как внешний вид документа (форматирование символов, организации абзацев, таблиц и т.п.), так и связи с другими URL-ресурсами (HTML-страницами, изображениями, серверами).

Web-страница представляет собой обычный текстовый файл (то есть файл, содержащий только текст с символами, записанными в кодах ASCII –американский стандартный код по обмену информацией), в котором собственно текст страницы размечен инструкциями для Web-браузера. На основании этих инструкций браузер определенным образом отображает страницу в своем окне и при активизации гиперссылки осуществляет загрузку указанного ресурса.

Таким образом, для создания Web-страниц достаточно изучить HTML, иметь на своем компьютере любой текстовый редактор для создания HTML-документа и браузер для его тестирования.

Однако существует целый ряд возможностей разработки Web-страниц без непосредственного набора тегов HTML. К таким возможностям относятся:

- работа в специализированных HTML-редакторах;
- использование программ-конвертеров, преобразующих документы в формат HTML, например, приложений MS Office;
- использование многофункциональных систем разработки и публикации сайтов (MS Fron Page, Macromedia Dreamweaver).

HTML-редакторы – это текстовые редакторы, содержащие дополнительные средства для быстрого ввода команд HTML и проверки их правильности.

Программы-конвертеры позволяют преобразовать существующий документ из его формата в набор Web-страниц. Приложения MS Office имеют встроенные средства преобразования своих документов в формат HTML.

Многофункциональные системы обладают комплексом инструментальных средств для создания и публикации Web-сайта. Они позволяют в процессе редактирования страницы видеть ее так, как она будет отображена в браузере, то есть реализуют принцип WYSIWYG («WhatYouSeeIsWhatYouGet» – «Что видите, то и получаете»). В таких системах HTML-текст создается автоматически.

HTML оказал существенное влияние на развитие INTERNET, однако часть ограничений этого языка разметки документов, ориентированного на представление документов в браузерах, служат препятствием для реализации

задач электронного бизнеса. Сегодня для бизнеса важно обрабатывать, перестраивать, хранить, обмениваться информацией, кодировать и подписывать документы. Решение этих проблем связывают с использование языка XML (ExtensibleMarkupLanguage– расширяемый язык разметки).

У XML много преимуществ перед HTML. К ним относятся высокий уровень отображения в браузере, возможность более гибкого создания структуры данных и использования информационных ресурсов, реализация механизма поиска и извлечения данных в документах.

Знакомство с основами HTML и XML, а также с некоторыми инструментальными средствами разработки Web-сайтов будет полезно любому маркетологу. Именно эти начальные знания позволят разговаривать профессиональным экономистам и программистам на одном языке. Это значит, что инвестиции фирмы в электронный маркетинг не будут напрасными: корпоративный Web-сайт будет соответствовать стратегическим планам фирмы и задачам маркетинга и базироваться на грамотном техническом решении.

Тема 6. Основы программирования на JavaScript

JavaScript — это относительно простой объектно-ориентированный язык, предназначенный для создания небольших клиентских и серверных приложений для Internet. Программы, написанные на языке JavaScript, включаются в состав HTML-документов и распространяются вместе с ними. Программы просмотра (браузеры – от англ. browser) типа Netscape Navigator и Microsoft Internet Explorer распознают встроенные в текст документа программы-вставки (script-коды) и выполняют их. Таким образом, JavaScript — интерпретируемый язык программирования. Примерами программ на JavaScript могут служить программы, проверяющие введенные пользователем данные или выполняющие какие-то действия при открытии или закрытии документа. Такие программы могут реагировать на действия пользователя — нажатие кнопок "мыши", ввод данных в экранной форме или перемещение "мыши" по странице. Более того, JavaScript-программы могут управлять самим браузером и атрибутами документа.

Язык JavaScript, будучи схожим по синтаксису с языком Java, за исключением объектной модели, в то же время не обладает такими свойствами, как статические типы данных и строгой типизацией. В JavaScript, в отличие от Java, понятие классов не является основой синтаксических конструкций языка. Такой основой является небольшой набор предопределенных типов данных, поддерживаемых исполняемой системой: числовые, булевские и строковые; функции, которые могут быть

как самостоятельными, так и методами объектов (метод в терминологии JavaScript — не что иное, как функция/подпрограмма); объектная модель с большим набором предопределенных объектов со своими свойствами и методами, а также правилами задания в программе пользователя новых объектов.

JavaScript был создан в 1995 году как инструмент, предоставляющий веб-дизайнерам возможности программирования. Все современные браузеры имеют поддержку JavaScript. JavaScript встраивается прямо в веб-страницы и исполняется браузером во время их загрузки/

JavaScript добавляется на веб-страницы с помощью тэга <script>. В следующем примере на страницу выводится текст «Я изучаю JavaScript».

```
<script type="text/javascript">
document.write("Я изучаю JavaScript.");
</script>
```

JavaScript код также может храниться во внешнем текстовом файле с расширением .js и подключаться к страницам с помощью тэга <script> следующим образом:

```
<script type="text/javascript" src="ex.js"></script>
```

JavaScript чувствителен к регистру букв.

В JavaScript существует два вида комментариев: короткие и длинные. Все, что находится после символа // до конца строки, будет являться коротким комментарием. Многострочные комментарии начинаются с /* и заканчиваются */.

В JavaScript предусмотрены три стандартных метода для ввода и вывода данных: **alert()**, **prompt()** и **confirm().**

Метод **alert()** позволяет выводить диалоговое окно (окно оповещения) с заданным сообщением и кнопкой ОК. Используется в случаях, когда необходимо, чтобы пользователь обязательно обратил внимание на определенную информацию.

Метод **confirm()** позволяет вывести диалоговое окно (окно подтверждения) с сообщением и двумя кнопками – ОК и Отмена (Cancel). Используется в случаях, когда необходимо, чтобы пользователь подтвердил или отклонил что-либо. Если пользователь щелкнул на кнопке ОК, то возвращается значение true; если щелкнул на кнопке Отмена, то возвращается значение false.

Метод **prompt()** позволяет вывести на экран диалоговое окно (окно запроса) с сообщением, а также с текстовым полем, в которое пользователь может ввести данные. Используются в случаях, когда от пользователя

необходимо получить определенную информацию. В этом окне предусмотрены две кнопки: ОК и Отмена (Cancel). В отличие от методов alert() и confirm() данный метод принимает два параметра: сообщение и значение, которое должно появиться в текстовом поле ввода данных по умолчанию.

Пример применения методов для ввода и вывода данных:

```
<html>
<head>
<script type='text/javascript'>
function fun1() {
   alert('Я окно оповещения');
}
function fun2() {
   confirm('Я окно подтверждения');
}
function fun3() {
   x=prompt('Введите Ваше имя:', 'Имя');
   document.write('Здравствуйте, '+ x);
   }
</script>
</head>
<body>
<input type='button' value='Окно оповещения'
onclick='fun1()' />
<input type='button' value='Окно подтверждения'
onclick='fun2()' />
<input type='button' value='Окно запроса' onclick='fun3()'
/>
</body>
</html>
```


Типы данных. Типы данных представлены в таблице 1.

Таблица 1. – Типы данных в Java Script

Тип данных	Примеры	Описание значений
Строковый или символьный (string)	"Привет" "д.т. 123-4567"	Последовательность символов, заключенная в кавычки, двойные или одинарные
Числовой (number)	3.14 -567 +2.5	Число, последовательность цифр, перед которой может быть указан знак числа (+ или перед положительными числами не

		обязательно ставить знак «+»; целая и дробная части чисел разделяются точкой. Число записывается без кавычек
Логический (булевский, boolean)	true false	true (истина, да) или false (ложь, нет); возможны только два значения
Null		Отсутствие какого бы то ни было значения
Объект (object)		Программный объект, определяемый своими свойствами. В частности, массив также является объектом
Функция (function)		Определение функции – программного кода, выполнение которого может возвращать некоторое значение

Арифметические операторы. Арифметические операторы используются для выполнения арифметических операций над переменными или значениями.

Таблица 2. – Арифметические операторы

Оператор	Название	Пример
+	Сложение	X + Y
-	Вычитание	X – Y
*	Умножение	X * Y
/	Деление	X / Y
%	Деление по модулю	X % Y
++	Увеличение на 1	X++
--	Уменьшение на 1	Y--

Чтобы уменьшить размер кода, можно использовать сокращенную запись арифметических операций. Например, вместо x= x + y писать x+ = y.

Операторы сравнения. Операторы сравнения представлены в таблице.

Таблица 3. – Операторы сравнения.

Оператор	Название	Пример
==	Равно	X==Y
!=	Не равно	X!=Y
>	Больше, чем	X>Y
>=	Больше или равно	X>=Y
<	Меньше, чем	X<Y
<=	Меньше или равно	X <= Y

Оператор «равно» записывается с помощью двух символов без пробелов между ними.

Логические операторы. Логические данные, обычно получаемые с помощью элементарных выражений, содержащих операторы сравнения, можно объединять в более сложные выражения. Для этого используются логические (булевские) операторы: И (&&), ИЛИ (||), НЕ (!). Выражения с логическими операторами возвращают значение true или false.

Операторы условного перехода. Оператор условного перехода **if** позволяет реализовать структуру условного выражения если ..., то ..., иначе ...

Пример оператора if перехода:

```
if (sr_ball>=5)
{document.write("Стипендия есть");
}
else
{document.write("Стипендии нету");
}
```

В фигурных скобках располагается блок кода – несколько выражений. Если в блоке используется одна команда, то фигурные скобки можно не писать.

Оператор **Switch** (переключатель) удобно использовать, если требуется проверить несколько условий, которые не являются взаимоисключающими.

Пример оператора switch:

```
switch (sr_ball)
{
case 5: document.write("Стипендия минимальная")
case 6: document.write("Стипендия с коэффициентом 1,2")
case 9: document.write("Стипендия с коэффициентом 1,4")
}
```

Функции. Функция представляет собой подпрограмму, которую можно вызвать для выполнения, обратившись к ней по имени. Взаимодействие функции с внешней программой, из которой она была вызвана, происходит путем передачи функции параметров и приема от нее результата вычислений. Функция в JavaScript может и не требовать параметров, а также ничего не возвращать.

В JavaScript есть встроенные функции, которые можно использовать в программах, но код которых нельзя редактировать или посмотреть.

Пользователь также может создать свои функции для решения конкретных задач – пользовательские.

Описание функции имеет следующий синтаксис:

```
function имя_функции(параметры){
код
}
```

Функция **messageWrite()** в примере ниже будет выполнена только после нажатия на кнопку. В примере используется событие **onclick**.

```
<html>
<head>
<script type='text/javascript'>
// Функция выводит текст на страницу
function messageWrite() {
   document.write('Текст выводится на страницу с
помощью JavaScript!');
}
</script>
</head>
<body>
<input type='button' value='Нажми на меня'
onclick='messageWrite()' />
</body>
</html>
```

Операторы цикла. Цикл – это блок команд, который может повторно выполняться, пока определенное условие не будет выполнено.

JavaScript поддерживает 3 вида циклов: **for**, **while**, **do… while**.

Оператор for. Цикл for исполняет блок команд, пока заданное условие является истинным. Например,

```
for (i=1;i<=30;i++) {
   document.write (i+'<br />');
}
```

Оператор while. Цикл while выполняет блок кода, пока заданное условие истинно. Напрмер,

```
var i=1;
while (i<=30) {
   document.write (i+'<br />');
```

```
    i++;
}
```

Оператор do-while. Цикл do…while часто называют циклом с постусловием, потому что в отличие от предыдущих циклов он вначале исполняет блок команд и только потом проверяет заданное условие. Если условие истинно, блок команд выполняется еще раз, если условие ложно, цикл завершает исполнение.

В отличие от оператора while в операторе do-while код выполняется хотя бы один раз, независимо от условия. В примере код выполнится один раз, так как условие ложно.

```
var i=20;
do {
   document.write('Я студент (от лат. studens – усердно работающий, занимающийся) БГТУ');
}
while (i<=3);
```

Объекты. Объекты представляют собой программные единицы, обладающие некоторыми свойствами. Об объекте можно судить по значениям его свойств и описанию того, как он функционирует.

Встроенные объекты имеют фиксированные названия и свойства. Все свойства этих объектов разделяют на два вида: просто свойства и методы. Свойства аналогичны обычным переменным. Они имеют имена и значения. Некоторые свойства объектов доступны только для чтения, их значения нельзя изменять. Другие свойства доступны и для записи – их значения можно изменять с помощью оператора присваивания. Методы аналогичны функциям, они могут иметь параметры или не иметь их.

Таким образом, объект можно понимать как некоторый контейнер, содержащий переменные-свойства и функции-методы.

Для разработчиков веб-сайтов особенно важны объекты String (обработка строк), Array (массивы), Math (математические формулы и константы) и Date (работа с датами).

Объект Math. Встроенный объект **Math** позволяет производить математические операции. Чтобы обращаться к свойствам и методам этого объекта его не нужно предварительно создавать (в отличие от остальных встроенных объектов JavaScript).

Свойства объекта **Math** содержат значения часто используемых математических констант. В примере выводятся на страницу некоторые константы.

```
//Выведем на страницу число Пи
document.write(Math.PI + '<br />');
//Выведем экспоненту
document.write(Math.E + '<br />');
//Выведем натуральный логарифм 10
document.write(Math.LN10 + '<br />');
//Выведем квадратный корень 2
document.write(Math.SQRT2);
```

С помощью методов объекта можно производить над числами различные математические операции. Ниже перечислены методы **Math**:

- abc(число) – возвращает модуль (абсолютное значение) числа;
- acos(число) – возвращает арккосинус числа;
- asin(число) – возвращает арксинус числа;
- atan(число) – возвращает арктангенс числа;
- atan2(x, y) – возвращает угол в полярных координатах точки;
- ceil(число) – округляет число вверх до ближайшего целого;
- cos(число) – возвращает косинус числа;
- exp(число) – возвращает число **e** в степени число;
- floor(число) – округляет число вниз до ближайшего целого;
- log(число) – возвращает натуральный логарифм числа;
- max(число1, число2) – возвращает большее из 2 чисел;
- min(число1, число2) – возвращает меньшее из 2 чисел;
- pow(число1, число2) – возвращает число1 в степени число2;
- random () – возвращает случайное число между 0 и 1;
- round(число) – округляет число до ближайшего целого;
- sin(число) – возвращает синус числа;
- sqrt(число) – возвращает квадратный корень из числа;
- tan(число) – возвращает тангенс числа.

В примере демонстрируется применение методов объекта **Math**. Все результаты выводятся на страницу.

```
<html>
<body>
<script type='text/javascript'>
//Округлим число 25.34 до ближайшего целого
document.write(Math.round(25.34) + '<br />');
//Округлим число 25.88 до ближайшего целого
```

```
document.write(Math.round(25.88));
//Выберем из чисел 10 68 35 12 44 максимальное
document.write(Math.max(10,68,35,12,44)   +   '<br
/>');
//Выберем из чисел 10 68 35 12 44 минимальное
document.write(Math.min(10,68,35,12,44));
//Возведем 5 в -1 степень
document.write(Math.pow(5,-1));
//Сгенерируем случайное число между 0 и 1
document.write(Math.random() + '<br />');
//Сгенерируем случайное число между 0 и 100 и
округлим
document.write(Math.round(Math.random()*100));
</script>
</body>
</html>
```

Объект Number (Число). В JavaScript числа могут быть двух типов: целые и с плавающей точкой. Числа можно создавать обычным образом с помощью переменных и оператора присваивания, не прибегая к объекту Number. Однако этот объект обладает некоторыми полезными свойствами и методами, которые иногда могут пригодиться.

Пример создания объекта Number:

```
x = new Number('34.21').
```

Методы Number:

– **toExponential (количество)** – представляет число в экспоненциальной форме, параметр *количество* – целое число, определяющее, сколько цифр после точки следует указывать.

– **toFixed (количество)** – представляет число в форме с фиксированным количеством цифр после точки, параметр *количество* – целое число, определяющее, сколько цифр после точки следует указывать.

– **toPrecision (точность)** – представляет число с заданным общим количеством значащих цифр. Параметр *точность* – целое число, определяющее, сколько всего цифр, до и после точки, следует указывать.

– **toString (основание)** – возвращает строковое представление числа в системе счисления с указанным основанием. Если параметр не указан, имеется в виду десятеричная система счисления. Этот метод имеют все объекты.

Например, следующий код выведет на страницу число 3.422e+1:

```
x = new Number(34.215);
x=x.toExponential(3);
document.write(x).
```

Объект Array (Массив). Массив представляет собой упорядоченный набор данных. Его удобно представить себе в виде одностолбцовой таблицы, содержащей некоторое количество строк. В ячейках такой таблицы могут находиться данные любого типа, в том числе и массивы.

Можно создать массивы тремя разными способами.

Первый способ:

```
spec = new Array();
spec[0] = ИСиТ;
spec[1] = "ПОИТ";
spec[2] = "ПОИБМС";
spec[3] = "ДЭВИ";
```

Второй способ:

```
var spec = new Array("ИСиТ","ПОИТ","ПОИБМС","ДЭВИ");
```

Третий способ:

```
var spec = [("ИСиТ","ПОИТ","ПОИБМС","ДЭВИ")];
```

Нумерация индексов в массивах начинается не с 1, а с 0. С помощью свойства **length** можно узнать количество элементов в массиве.

Методы Array:

– **concat ()** объединяет два и более массива в один;

– **pop ()** – удаляет последний элемент массива и возвращает его значение;

– **push (значение|объект)** – добавляет к массиву указанное значение в качестве последнего элемента и возвращает новую длину массива;

– **shift ()** – удаляет первый элемент массива и возвращает его значение;

– **slice (индекс1 [, индекс2])** – создает массив из элементов исходного массива с индексами указанного диапазона;

– **sort ([функция_сортировки])** – сортирует (упорядочивает) элементы массива с помощью функции сравнения.

Объект String (Строка). Объект String (строковый объект) используется для хранения и обработки текстовой информации.

Примеры создания:

`mystring = new String ("Привет!")` – первый способ;

`mystring = "Привет!"` – второй способ.

С помощью свойства **length** можно узнать длину строки:

```
document.write(mystring.length)
```

С помощью метода **toUpperCase()** можно перевести все символы текста в верхний регистр, а с помощью **toLowerCase()** – в нижний.

Метод **concat()** позволяет объединить две и более строки и вывести результат на страницу:

```
str='Кафедра';
document.write(str.concat('ИСиТ'));
```

Метод **replace()** позволяет заменить одно произвольное слово в строке на другое:

```
document.write(str.replace('ИСиТ','ПОИТ'));
```

Объект Date (Дата). Объект **Date** позволяет производить различные операции с датой и временем. Некоторые методы объекта представлены в таблице.

Таблица 4 – Объект Дата

Метод	Описание
getDate()	Возвращает день месяца (может принимать значения от 1-31) заданной даты.
getDay()	Возвращает день недели (может принимать значения от 0-6, причем 0-Воскресенье, а 6-Суббота) заданной даты.
getFullYear()	Возвращает год (4 числа, например 2017) заданной даты.
getHours()	Возвращает час (может принимать значения от 0-23) заданной даты.
getMiliseconds()	Возвращает миллисекунду (может принимать значения от 0-999) заданной даты.
getMinutes()	Возвращает минуту (может принимать значения от 0-59) заданной даты.
getMonth()	Возвращает месяц (может принимать значения от 0-11) заданной даты.
getSeconds()	Возвращает секунду (может принимать значения от 0-59) заданной даты.
toDateString()	Преобразует часть объект, содержащую дату, в строку.
toString()	Преобразует объект в строку.

toTimeString()	Преобразует часть объекта, содержащую время, в строку.

После того, как объект создан, можно с помощью доступных методов производить над ним различные операции.

Пример.

```
//Создадим объект Date
x=new Date();
//Извлечем день месяца и выведем результат на страницу
document.write(x.getDate());
//Извлечем год из объекта x и выведем результат на страницу
document.write(x.getFullYear());
```

Пользовательские объекты в JavaScript можно создать несколькими способами.

Один из способов основан на функции, в теле которой описываются все свойства и методы создаваемого объекта. Ее называют функцией-конструктором или просто конструктором объекта. Имя функции-конструктора объекта является одновременно и именем создаваемого объекта. Свойства и методы создаваемого объекта задаются в теле функции-конструктора с помощью операторов присваивания, имена переменных-свойств записываются с ключевым словом **this** (этот).

Пример создания и использования объекта, созданного с помощью конструктора.

```
//Конструктор для создания объекта Gruppa со свойствами n, spec, kolich и методом add_stud
function Gruppa(n,spec,kolich) {
//Свойства(номер, специальность, количество)
this.n=n;
this.spec=spec;
this.kolich=kolich;
//Метод (добавить в группу k студентов)
this.add_stud=function add_stud(k) {
                this.kolich+=k;
document.write('Вгруппу ' + this.n + ' добавили'+ k + ' студентов.<br\>');
}
   }
```

```
      //Теперь можно создавать экземпляры объекта
gr1=newGruppa(2, 'ИСиТ', 28);
      //Вызов метода созданного объекта (добавить 2
студента)
      gr1.add_stud(2);
      //С помощью prototype добавим объекту свойство
kurs,
      //экземпляры объекта будут иметь это свойство
      Gruppa.prototype.kurs=this.kurs
      gr1.kurs=2
      document.write(gr1.n,gr1.spec,          gr1.kolich,
gr1.kurs)
      </script>
```

Обращаться к свойствам объектов можно двумя способами:

– используя точку после имени объекта, например **gr1.kurs=2**;

– заключая название свойства в квадратные скобки после имени объекта, например **gr1.[‘kurs’]=2**.

С помощью свойства **prototype** можно добавлять новые свойства и методы к конструкторам объектов. Добавленные к конструктору свойства и методы будут также добавлены ко всем объектам, которые были созданы данным конструктором.

Например, `Gruppa.prototype.kurs=this.kurs`.

Удаление свойств объекта (delete). С помощью оператора **delete** можно удалить свойство объекта, а также элемент массива. При удалении элемента массива удаляется и его индекс, но оставшиеся элементы сохраняют свои прежние индексы, а длина массива не изменяется.

Пример, delete mas[2] – удалить 3-й элемент массива.

Проверка наличия свойств (in). Оператор **in** позволяет проверить, имеется ли некоторое свойство или метод у того или иного объекта. Если свойство или метод содержится в объекте, то возвращается true, иначе – false. Отсюда следует, что оператор in можно применять в условных выражениях (в операторах if, switch, for, while, do-while).

Примеры:

```
document.write('spec' in gr1) // проверить, есть
ли свойство spec у объекта gr1;
document.write(1 in mas) //проверить, есть ли
элемент с номером 1 в массиве mas.
```

Проверка принадлежности объекта модели (instance of). Оператор **instance of** позволяет проверить, принадлежит ли некоторый объект объектной модели JavaScript. Если они совпадают, метод возвращает true, если нет false.

Выражение с оператором instance of может использоваться в условных выражениях (в операторах if, switch, for, while, do-while).

Пример, document.write(mas instance of Array) – проверяет, является ли mas массивом.

Определение типа (type of). Оператор **type of** позволяет проверить, относится ли значение к одному из следующих типов: string, number, boolean, object, function или undefined. Значение, возвращаемое оператором type of, является строковым и содержит одно из перечисленных названий типа.

Пример, `document.write(typeof gr1.kurs).`

Для создания программ на JavaScript не требуется никаких дополнительных средств— необходим лишь браузер, поддерживающий язык JavaScript соответствующей версии и текстовый редактор, позволяющий создавать HTML-документы. Так как программа на JavaScript встраивается непосредственно в текст HTML-документа, вы можете немедленно увидеть результаты своей работы во время просмотра документа браузером и при необходимости внести изменения.

Тема 7. Основы программирования на PHP

Язык PHP был разработан как инструмент для решения чисто практических задач. Его создатель, Расмус Лердорф, хотел знать, сколько человек читают его online-резюме, и написал для этого простенькую CGI-оболочку на языке Perl, т.е. это был набор Perlскриптов, предназначенных исключительно для определенной цели – сбора статистики посещений.

CGI (Common Gateway Interface – общий интерфейс шлюзов) является стандартом, который предназначен для создания серверных приложений, работающих по протоколу HTTP. Такие приложения (их называют шлюзами или CGI-программами) запускаются сервером в режиме реального времени. Сервер передает запросы пользователя CGI-программе, которая их обрабатывает и возвращает результат своей работы на экран пользователя. Таким образом, посетитель получает динамическую информацию, которая может изменяться в результате влияния различных факторов. Сам шлюз

(скрипт CGI) может быть написан на различных языках программирования – Си/C++, Fortran, Perl, TCL, UNIX Shell, Visual Basic, Python и др.

Вскоре выяснилось, что оболочка обладает небольшой производительностью, и пришлось переписать ее заново, но уже на языке Си. После этого исходники были выложены на всеобщее обозрение для исправления ошибок и дополнения. Пользователи сервера, где располагался сайт с первой версией PHP, заинтересовались инструментом, появились желающие его использовать. Так что скоро PHP превратился в самостоятельный проект, и в начале 1995 года вышла первая известная версия продукта, называвшаяся Personal Home Page Tools (средства для персональной домашней страницы). Средства эти были более чем скромными: анализатор кода, понимающий всего лишь несколько специальных команд, и набор утилит, полезных для создания гостевой книги, счетчика посещений, чата и т.п.

В настоящее время ведутся работы по улучшению Zend Engine и внедрению нововведений в PHP 5.0, первые бета-версии которого уже вышли в свет. Одно из существенных изменений произошло в объектной модели языка, ее основательно подлатали и добавили много новых возможностей.

Сегодня PHP используется сотнями тысяч разработчиков. Несколько миллионов сайтов написаны на PHP, что составляет более 20% доменов Internet.

В первую очередь PHP используется для создания скриптов, работающих на стороне сервера, для этого его, собственно, и придумали. PHP способен решать те же задачи, что и любые другие CGI-скрипты, в том числе обрабатывать данные html-форм, динамически генерировать html страницы и т.п. Но есть и другие области, где может использоваться PHP. Всего выделяют три основные области применения PHP.

Первая область, как уже говорилось, – это создание приложений (скриптов), которые исполняются на стороне сервера. PHP наиболее широко используется именно для создания такого рода скриптов. Для того чтобы работать таким образом, понадобится PHP-парсер (т.е. обработчик php-скриптов) и web-сервер для обработки скрипта, браузер для просмотра результатов работы скрипта, ну, и, конечно, какой-либо текстовый редактор для написания самого php-кода. Парсер PHP распространяется в виде CGI-программы или серверного модуля. Как установить его и web-сервер на свой компьютер, мы рассмотрим немного позднее. В этом курсе мы будем обсуждать, как правило, создание именно серверных приложений, как пример использования языка PHP.

Вторая область – это создание скриптов, выполняющихся в командной строке. То есть с помощью PHP можно создавать такие скрипты, которые

будут исполняться, вне зависимости от web-сервера и браузера, на конкретной машине. Для такой работы потребуется лишь парсер PHP (в этом случае его называют интерпретатором командной строки (cli, command line interpreter)). Этот способ работы подходит, например, для скриптов, которые должны выполняться регулярно с помощью различных планировщиков задач или для решения задач простой обработки текста.

И последняя область – это создание GUI-приложений (графических интерфейсов), выполняющихся на стороне клиента. В принципе это не самый лучший способ использовать PHP, особенно для начинающих, но если вы уже досконально изучили PHP, то такие возможности языка могут оказаться весьма полезны. Для применения PHP в этой области потребуется специальный инструмент – PHP-GTK, который является расширением PHP.

Если говорить о возможностях сегодняшнего PHP, то они выходят далеко за рамки тех, что были реализованы в его первых версиях. С помощью PHP можно создавать изображения, PDF-файлы, флэш-ролики, в него включена поддержка большого числа современных баз данных, встроены функции для работы с текстовыми данными любых форматов, включая XML, и функции для работы с файловой системой. PHP поддерживает взаимодействие с различными сервисами посредством соответствующих протоколов, таких как протокол управления доступом к директориям LDAP, протокол работы с сетевым оборудованием SNMP, протоколы передачи сообщений IMAP, NNTP и POP3, протокол передачи гипертекста HTTP и т.д.

Элементы языка: константы и выражения; функции; классы; операторы; регулярные выражения. Константы и выражения: любой скрипт PHP состоит из последовательности операторов. Оператор может быть присваиванием, вызовом функции, циклом, условным выражением или пустым выражением. Операторы обычно заканчиваются точкой с запятой. Также операторы могут быть объединены в группу заключением группы операторов в фигурные скобки. Группа операторов также является оператором. PHP определяет несколько констант и предоставляет механизм для определения Ваших констант. Константы похожи на переменные, но они имеют иной синтаксис.

Константы. Предопределенные константы - это __FILE__ and __LINE__, которые соответствуют имени файла и номеру строки, которая выполняется в настоящий момент. Можно определить дополнительные константы с помощью функций define() и undefine(). Арифметические операции: $a + $b – сложение – сумма $a и $b; $a - $b – вычитание – вычитает $b из $a; $a * $b – умножение – произведение $a и $b; $a / $b – деление – деление $a на $b; $a % $b – остаток деления – остаток от деления $a на $b. Оператор деления ("/") возвращает целую величину (результат

целочисленного деления) если оба оператора - целые (или строка преобразованная в целое). Если каждый операнд является величиной с плавающей запятой, выполнится деление с плавающей запятой.

Операторы строк и присваивания. Операторы строк: в действительности есть только один оператор - конкатенации (".").

```
$a = "Hello";
$b = $a . "World!";
```

Операторы присваивания: основным оператором присваивания является "=".

```
$a = ($b = 4) + 5;
```

Есть дополнительные "операторы с присваиванием" для всех арифметических и строковых операторов:

```
$a = 3; $a += 5; $b = "Hello ";
$b .= "There!";
```

Бинарные операторы:

$a & $b	И	$a=5; /* 0101 */ $b=12; /*1100*/ $c=$a & $b; /* 4 (0100) */
$a | $b	Или	$a=5; /*0101*/ $b=12;/*1100*/ $c=$a|$b;/*1101*/
~ $a	Не	$a=5; /* 0101 */ ~ $a; /*1010 */

Логические операторы:

$a and $b	И	Истина, если истинны $a и $b.
$a or $b	Или	Истина, если истинны $a или $b.
$a xor $b	Или	Истина, если истинны $a или $b, но не оба.
! $a	Не	Истина, если не истинно $a.
$a && $b	И	Истина, если истинны и $a и $b.
$a || $b	Или	Истина, если истинны $a или $b.

Разница двух различных вариантов операторов "and" и "&&" - в различии приоритетов.

Операторы сравнения:

$a = = $b	истина, если $a эквивалентно $b.
$a != $b	Истина, если $a не эквивалентно $b.
$a < $b	Истина если $a меньше чем $b.
$a > $b	Истина если $a больше $b.
$a <= $b	Истина, если $a меньше или равно $b.
$a >= $b	Истина, если $a больше или равно $b.

Выражения. В PHP почти все является выражениями. Простейший пример, приходящий на ум - это константы и переменные. Когда вы печатаете "$a = 5", вы присваиваете значение '5' переменной $a. После этого присваивания вы считаете значением $a 5, также, если вы напишете $b = $a, вы будете ожидать того же как, если бы вы написали $b = 5. Другими словами, $a это также выражение со значением 5. Запись типа '$b = ($a = 5)' похожа на запись '$a = 5; $b = 5;' (точка с запятой отмечает конец выражения). Так как присваивания рассматриваются справа налево, вы также можете написать '$b = $a = 5'.

Более сложные примеры выражений - это функции: `function foo () { return 5; }`. Функции - это выражения с тем значением, которое они возвращают. Так как foo() возвращает 5, значение выражение 'foo()' - 5. PHP поддерживает 3 скалярных типа значений: целое, число с плавающей точкой и строки. PHP поддерживает 2 составных (нескалярных) типа: массивы и объекты. Каждое из таких значений может быть присвоено переменной или возвращено функцией.

Во многих случаях, в основном в условных операторах и операторах циклов, важно, являются ли их значения TRUE или FALSE (в PHP нет специального типа boolean). Любое не нулевое целое значение - это TRUE, ноль - это FALSE. Обратите внимание на то, что отрицательные значения - это не ноль и поэтому они считаются равными TRUE. Пустая строка и строка '0' это FALSE; все остальные строки - TRUE. И насчет составных типов (массивы и объекты) - если значение такого типа не содержит элементов, то оно считается равным FALSE; иначе подразумевается TRUE.

IF. Возможности PHP по использованию выражения IF похожи на C: `if (expr) statement`. Вычисляется логический результат "expr". Если expr равно TRUE, то PHP выполнит "statement", а если FALSE - проигнорирует. `if ($a > $b) { print "a is bigger than b"; $b = $a; }`. Выражение IF может иметь неограниченную степень вложенности в другие выражения IF, что позволяет Вам использовать выполнение по условию различных частей программы.

ELSE. ELSE расширяет возможности IF по части обработки вариантов выражения, когда оно равно FALSE. Данный пример выведет фразу 'a is bigger than b' если $a больше $b, и 'a is NOT bigger than b' в противном случае: `if ($a>$b){print "a is bigger than b";} else {print "a is NOT bigger than b"; }`. Выражение ELSE выполняется только если выражение IF равно FALSE, а если есть конструкции ELSEIF - то если и они также равны FALSE.

ELSEIF. Является комбинацией IF и ELSE. ELSEIF, как и ELSE позволяет выполнить выражение, если значение IF равно FALSE, но в

отличие от ELSE, оно выполнится только если выражение ELSEIF равно TRUE: `if ($a>$b){print "a is bigger than b";} elseif ($a= =$b){print "a is equal b";} else { print "a is smaller than b"; }`. Внутри одного выражения IF может быть несколько ELSEIF. Первое выражение ELSEIF (если таковые есть), которое будет равно TRUE, будет выполнено. В PHP вы можете написать 'else if' (два слова), что будет значить то же самое, что и 'elseif' (одно слово).

IF(): ... ENDIF. Наиболее часто это используется, когда вы внедряете блоки HTML внутрь оператора IF. Вместо использования фигурных скобок за "IF(выражение)" должно следовать двоеточие, одно или несколько выражений и завершающий ENDIF: A = 5. Блок HTML будет виден, если $a равно 5. Этот альтернативный синтаксис применим и к ELSE и ELSEIF (expr) : `if ($a == 5): print "a equals 5"; elseif ($a == 6): print "a equals 6"; else: print "a is neither 5 nor 6"; endif;`.

WHILE(expr) statement - предписывает выполнять statement до тех пор, пока expr равно TRUE. Значение expr проверяется в начале цикла, и если значение expr изменится внутри цикла, то он не прервется до конца текущей итерации. Если изначально expr равно FALSE, цикл не выполняется ни разу. Можно сгруппировать несколько операторов внутри фигурных скобок или использовать альтернативный синтаксис: WHILE(expr): выражения ... ENDWHILE; Следующие примеры идентичны: `$i = 1; while ($i <= 10) {print $i++;} $i = 1;while ($i<= 10): print $i; $i++; endwhile;`.

В DO…WHILE значение логического выражения проверяется не до, а после окончания итерации и он гарантировано выполнится хотя бы один раз. Для циклов DO..WHILE существует только один вид синтаксиса: `$i = 0; do { print $i; } while ($i>0);`.

FOR(expr1; expr2; expr3) statement. expr1 безусловно вычисляется в начале цикла. В начале каждой итерации вычисляется expr2. Если оно равно TRUE, то выполняются statement. Если FALSE, то цикл заканчивается. В конце каждой итерации вычисляется expr3. Каждое из этих выражений может быть пустым. Если expr2 пусто, то цикл продолжается бесконечно. Например:

```
for ($i = 1; $i <= 10; $i++) print $i;
for ($i = 1;;$i++) {if ($i > 10) break;
    print $i;}.
```

PHP также поддерживает альтернативный синтаксис FOR: FOR (expr1; expr2; expr3): выражение; ...; endfor;. Другие языки используют оператор

foreach для того, чтобы обрабатывает массивы или списки. PHP использует для этого оператор while и функции list() и each().

BREAK. Прерывает выполнение текущего цикла.

```
$i = 0; while ($i < 10) {
    if ($arr[$i] == "stop") { break; }
    $i++; }
```

CONTINUE. Переходит на начало ближайшего цикла.

```
while (list($key,$value) = each($arr)) {
    if ($key % 2) {continue; }
    do_something_odd ($value); }
```

Оператор SWITCH похож на группу операторов IF с одинаковым выражением.

```
switch ($i) {
    case 0: print "i = 0"; break;
    case 1: print "i = 1"; break;
    case 2: print "i = 2"; break;
    default: print "i!= 0, 1 or 2";
}
```

Выражения в CASE могут быть любого скалярного типа, то есть целые числа или числа с плавающей запятой, а так же строки. Массивы и объекты не будут ошибкой.

Оператор REQUIRE заменяет себя содержимым указанного файла, похоже на то, как в препроцессоре C работает #include. `require ('header.inc');`.

Оператор INCLUDE вставляет и выполняет содержимое указанного файла. Это случается каждый раз, когда встречается оператор INCLUDE, так что вы можете включить этот оператор внутрь цикла, чтобы включить несколько файлов:

```
$files = array ('first.inc', 'second.inc', 'third.inc');
for ($i = 0; $i < count($files); $i++) { include($files[$i]); }
```

include() отличается от require() тем, что include выполняется каждый раз при его встрече, а require() заменяется на содержимое указанного файла безотносительно будет ли выполнено его содержимое или нет. include() - специальный оператор, его требуется заключать в фигурные скобки при использовании внутри условного оператора:

```
if ($condition) { include($file);}
else { include($other); }
```

Когда файл исполняется, парсер пребывает в "режиме HTML", то есть будет выводить содержимое файла, пока не встретит первый стартовый тег PHP (<?).

Функция может быть объявлена так:

```
function foo ($arg_1,..., $arg_n) {
    echo "Example function.\n";
    return $retval; }
```

Внутри функции может быть объявление другой функции или класса. Функции должны быть определены перед тем, как на них ссылаться. Результаты возвращаются через необязательный оператор return. Возвращаемый результат может быть любого типа, включая списки и объекты. `function my ($num) {return $num*$num; } echo my (4); // outputs '16'.`

Множественные результаты не могут быть возвращены в качестве результата, но вы можете реализовать это путем возврата списка:

```
function foo() {
    return array (0, 1, 2); }
list ($zero, $one, $two) = foo();
```

Информация может быть передана функции через список аргументов, которые являются разделенным запятыми списком переменных и/или констант. PHP поддерживает передачу аргументов по значению (по умолчанию), по ссылке, и значения по умолчанию.

Списки аргументов переменной длины не поддерживаются, но можно передать массивы:

```
function takes_array($input) {
    echo "$input[0] + $input[1] = ",
$input[0]+$input[1];}
```

По умолчанию аргументы передаются по значению. Если надо в функции модифицировать аргументы, передают их по ссылке, поставив амперсанд (&) перед именем аргумента в объявлении функции:

```
function foo( &$bar ) {
    $bar .= ' and something extra.'; }
$str = 'This is a string, ';
foo ($str); echo $str;
```

Функции могут определять значения по умолчанию для скалярных аргументов:

```
function makecoffee ($type = "cappucino") {
echo "Making a cup of $type.\n";
}
echo makecoffee ();
```

```
echo makecoffee ("espresso");
```

Этот пример выведет следующее: `Making a cup of cappucino. Making a cup of espresso.`

Значение по умолчанию должно быть константой, а не переменной или, к примеру, членом класса. Учтите, что когда вы объявляете аргументы по умолчанию, они должны быть справа от всех "неумолчиваемых" аргументов:

```
function makeyogurt ($flavour,
    $type = "acidophilus") {
        return "Making a bowl of
    $type $flavour.\n";
}
echo makeyogurt ("raspberry");
```

Классы. Вы должны создавать переменные класса, используя оператор new:

```
$cart = new Cart;
$cart->add_item("10", 1);
```

Классы могут быть расширениями других классов. Расширенный класс обладает всеми переменными и функциями базового класса и тем, что вы определите при расширении класса. Это делается, используя ключевое слово extends:

```
class Named_Cart extends Cart {
    var $owner;
function set_owner ($name) {
    $this->owner = $name; }
}
```

Внутри функций класса переменная $this означает сам объект. Вы должны использовать $this->нечто для доступа к переменной или функции с именем 'нечто' внутри объекта. Конструкторы - это функции в классе, которые автоматически вызываются, когда вы создаете новую переменную данного класса. Функция становится конструктором, когда она имеет такое же имя, как и сам класс.

```
class Auto_Cart extends Cart {
    function Auto_Cart () {
        $this->add_item ("10", 1); }
}
```

Конструкторы также могут иметь аргументы, и эти аргументы могут быть необязательными, что делает конструктор более полезным:

```
class Constructor_Cart {
```

```
    function Constructor_Cart ($item = "10", $num = 1)
{
        $this->add_item ($item, $num);
    }
}
$default_cart = new Constructor_Cart;
$different_cart = new Constructor_Cart ("20", 17);
```

Регулярные выражения. Регулярные выражения используются для сложного манипулирования строками в PHP. Функции, которые поддерживают регулярные выражения: – `ereg();` – `ereg_replace();` – `eregi();` – `eregi_replace();` – `split()`. Все эти функции принимают строку регулярного выражения как их первый параметр.

Примеры регулярных выражений:

– ereg("abc",$string); /* Возвращает 'истина', если "abc" найдено в $string. */

– ereg("^abc",$string);/* Возвращает 'истина', если "abc" найдено в начале $string. */

– ereg("abc$",$string);/* Возвращает 'истина', если "abc" найдено в конце $string.*/

– eregi("(ozilla.[23]|MSIE.3)",$HTTP_US ER_AGENT);/*Возвращает 'истина', если браузер клиента - Netscape 2, 3 или MSIE 3.*/

– ereg("([[:alnum:]]+) ([[:alnum:]]+) ([[:alnum:]]+)", $string,$regs);

– /* Помещает три слова - $regs[1], $regs[2] и $regs[3], разделенные пробелом. */

– ereg_replace("^","
",$string); /* Устанавливает тег
 в начало строки $string. */

– ereg_replace("$","
",$string); /* Устанавливает тег
 в конец строки $string. */

– ereg_replace("\n","",$string); – /* Отсекает символ "возврат каретки" в строке $string. */

Программа на PHP (да и на любом другом языке программирования) – это набор команд (инструкций). Обработчику программы (парсеру) необходимо как-то отличать одну команду от другой. Для этого используются специальные символы – разделители. В PHP инструкции разделяются так же, как и в Си или Perl, – каждое выражение заканчивается точкой с запятой.

Закрывающий тег «?>» также подразумевает конец инструкции, поэтому перед ним точку с запятой не ставят. Например, два следующих фрагмента кода эквивалентны:

```
<?php
echo "Hello, world!"; // точка с запятой
```

```
//в конце команды
//обязательна
?>
<?php
echo "Hello, world!" ?> <!-- точка с запятой
опускается из-за "?>" -->
```

На сегодняшний день PHP является наиболее распространенным языком веб-программирования. Подавляющие большинство сайтов и веб-сервисов в интернете написано с помощью PHP. По некоторым оценкам PHP применяется более чем на 80% сайтов, среди которых такие сервисы, как facebook.com, vk.com, baidu.com и другие. И такая популярность неудивительна. Простота языка позволяет быстро и легко создавать сайты и порталы различной сложности.

ПРАКТИЧЕСКИЙ РАЗДЕЛ

Методические указания к практическим и лабораторным работам

Темы практических и лабораторных занятий, предусмотренных в рамках дисциплины «Алгоритмизация и основы программирования», обусловлены программой дисциплины для студентов направления специальности 1 – 23 01 11 – 02 Библиотечно-информационная деятельность (автоматизация). Практические и лабораторные работы направлены на практическое закрепление теоретического материала, связанного с изучением разных форм организации данных в программах и методов их обработки при решении практических задач автоматизации; основными практическими приемами, методами и средствами анализа, построения и использования веб-программирования.

Предложенные для работы темы практических и лабораторных занятий взаимосвязаны и требуют от студентов последовательного изучения содержания дисциплины. Основными материалами, используемыми студентами в ходе подготовки к лабораторным занятиям, являются конспекты лекций и рекомендуемые преподавателем документные и электронные источники информации.

Практически и лабораторные работы в основном выполняются студентами на занятиях в компьютерной аудитории на персональном компьютере, а в отдельных случаях, в тетради или на листах, которые после выполнения задания сдаются на проверку преподавателю. Каждая работа оценивается по 10-балльной системе. При отсутствии студента на занятии, тема должна быть отработана.

Выполнение практических и лабораторных работ в полном объеме поможет студентам лучше подготовиться к итоговому рубежному контролю – зачету и экзамену. В процессе освоения учебной дисциплины «Алгоритмизация и основы программирования» возможно проведение со студентами индивидуальных консультаций.

Тематика и описание практических работ

№ п/п	Тема практической работы	Количество ауд. часов
1	Составление блок-схем и программ линейной структуры	2
2	Создание графики посредством элемента canvas	6
	Всего	**8**

ПРАКТИЧЕСКАЯ РАБОТА №1
па теме **«Составление блок-схем и программ линейной структуры»**, 2 часа

Цель: научиться составлять блок-схемы линейной и разветвляющей структуры; научиться определять результат выполнения алгоритма.

Алгоритм выполнения практической работы:

Задание 1. Создайте линейный алгоритм в виде блок-схемы для решения следующей задачи:

Вычислить площадь прямоугольника по заданной длине и ширине.

Для этого вам нужно внести следующие данные в элементы блок-схемы:

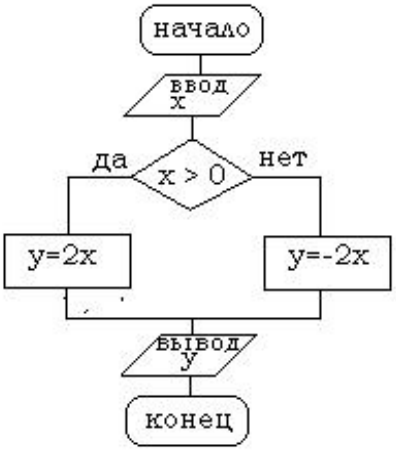
1) Ввести a и b.
2) Вычислить площадь S по формуле a*b.
3) Вывести полученный результат на экран.
4) Закончить выполнение алгоритма.

Задание 2. Составить блок-схему алгоритма вычисления периметра P и площади S квадрата со стороной длины A.

Задание 3. Составить блок-схему решения задачи нахождения значения функции z = y/x.

Задание 4. Дана блок-схема алгоритма (рисунок). Определить результат выполнения алгоритма при определенных значениях исходных данных, при x=16 и y=2.

Задание 5. Дана блок-схема алгоритма (рисунок). Определить результат выполнения алгоритма при определенных значениях исходных данных, при x=-6, x=0 и x=7.



Задание 6 (дополнительное). Придумайте свой собственный циклический алгоритм и изобразите его в виде блок-схемы.

Содержание отчета по работе: сохраненный на персональном компьютере файл.

Литература:

1. ГОСТ 19.701–90 (ИСО 5807–85) Единая система программной документации. Схемы алгоритмов, программ, данных и систем: обозначения условные и правила дополнения. – М.: Стандартинформ, 2010. – 158 с.

2. Алгоритм. Свойства алгоритма [Электронный ресурс]. – Режим доступа: https://pro-prof.com/archives/578. – Дата доступа: 06.12.2019.

3. Алгоритмы [Электронный ресурс]. – Режим доступа: https://pro-prof.com/books-algorithms. – Дата доступа: 06.12.2019.

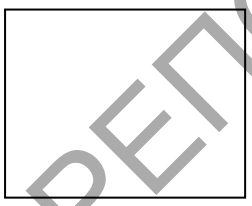
ПРАКТИЧЕСКАЯ РАБОТА №2
па теме **«Создание графики посредством элемента canvas»**, 6 часов

Цель: познакомиться с особенностями создания графических изображений посредством элемента canvas; изучить специфику использования языка программирования JavaScript при рисовании графики.

Алгоритм выполнения практической работы:

Часть 1. Задачи на работу с canvas на линии (*для решения задач данного блока используйте следующие методы: lineTo, moveTo, beginPath, closePath, stroke, fill).*

1. Нарисуйте на канвасе такую же фигуру, как показано в образце: —.
2. Нарисуйте на канвасе такую же фигуру, как показано в образце: |.
3. Нарисуйте на канвасе такую же фигуру, как показано в образце: X.
4. Нарисуйте на канвасе такую же фигуру, как показано в образце: □.
5. Нарисуйте на канвасе такую же фигуру, как показано в образце: ∇.
6. Нарисуйте на канвасе такую же фигуру, как показано в образце: →.
7. Используя методы **moveTo** и **lineTo** для рисования контура и метод **fill** для его заливки, нарисуйте на канвасе следующую фигуру:

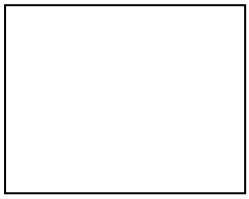


Часть 2. Задачи на работу с canvas на фигуры (*для решения задач данного блока используйте следующие методы: fillRect, strokeRect, rect).*

8. Используя метод **fillRect**, нарисуйте на канвасе следующую фигуру:

9. Используя метод **strokeRect**, нарисуйте на канвасе следующую фигуру:

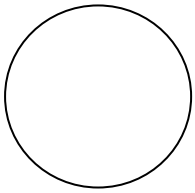
10. Используя метод **rect**, нарисуйте на канвасе следующую фигуру:



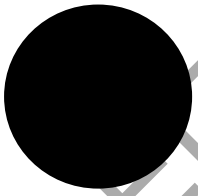
11. Используя метод **rect**, нарисуйте на канвасе следующую фигуру:

Часть 3. Задачи на работу с canvas на окружности (*для решения задач данного блока используйте следующие методы: arc)*.

12. Используя метод **arc**, нарисуйте на канвасе следующую фигуру:

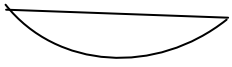


13. Используя метод **arc**, нарисуйте на канвасе следующую фигуру:



14. Используя метод **arc**, нарисуйте на канвасе следующую фигуру:

15. Используя метод **arc**, нарисуйте на канвасе следующую фигуру:



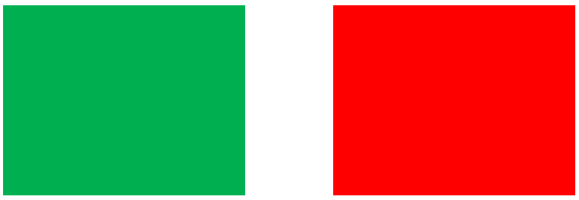
Часть 4. Задачи на работу с canvas на смену цвета (*для решения задач данного блока используйте следующие свойства: fillStyle, strokeStyle)*.

16. Используя цикл, нарисуйте на канвасе следующую фигуру:

17. Используя цикл, нарисуйте на канвасе следующую фигуру:

18. Используя цикл, нарисуйте на канвасе следующую фигуру:

19. Используя цикл, нарисуйте на канвасе следующую фигуру:



Содержание отчета по работе: сохраненный на персональном компьютере файл.

Литература:

1. Руководство по Canvas [Электронный ресурс]. – Режим доступа: https://developer.mozilla.org/ru/docs/Web/API/Canvas_API/Tutorial. – Дата доступа: 15.11.2019.

2. Canvas шаг за шагом: основы [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/111308/. – дата доступа: 15.11.2019.

Тематика и описание лабораторных работ

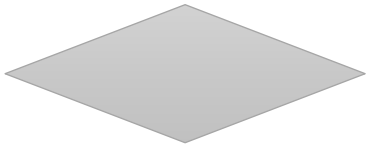
№ п/п	Тема лабораторной работы	Количество ауд. часов
1	Виды алгоритмов	2
2	Типы данных	4
3	Основные структуры данных	4
4	Основные структуры данных: функции	6
5	Управление процессом выполнения программы: условия	6
6	Управление процессом выполнения программы: циклы	6
	Всего	**28**

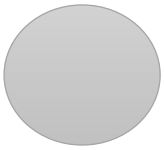
ЛАБОРАТОРНАЯ РАБОТА №1
па теме **«Виды алгоритмов»**, 2 часа

Цель: познакомиться с видами алгоритмов, научиться составлять блок-схемы на алгоритмические цепочки.

Алгоритм выполнения лабораторной работы:

1. Расшифруйте значения элементов блок-схемы алгоритма, заполнив таблицу.

2. Представьте в виде блок-схемы виды алгоритмов: линейный, циклический, алгоритм ветвления, комбинированный.

3. Раскройте основные правила проставления направления действий в алгоритмических конструкциях.

4. Приведите примеры из жизни, которые можно описать алгоритмом (линейным, циклическим, ветвления). Разберите примеры (типичные ошибки при составлении) с преподавателем.

Содержание отчета по работе: сохраненный на персональном компьютере файл.

Литература:

4. ГОСТ 19.701–90 (ИСО 5807–85) Единая система программной документации. Схемы алгоритмов, программ, данных и систем: обозначения условные и правила дополнения. – М.: Стандартинформ, 2010. – 158 с.

5. Алгоритм. Свойства алгоритма [Электронный ресурс]. – Режим доступа: https://pro-prof.com/archives/578. – Дата доступа: 06.12.2019.

6. Алгоритмы [Электронный ресурс]. – Режим доступа: https://pro-prof.com/books-algorithms. – Дата доступа: 06.12.2019.

ЛАБОРАТОРНАЯ РАБОТА №2
па теме **«Типы данных»**, 4 часа

Цель: познакомиться с основными типами данных в языках программирования: число, строка, булевый тип; изучить специфику использования типов данных в языке программирования JavaScript.

Алгоритм выполнения лабораторной работы:

Часть 1. Число.

1. Проанализируйте вместе с преподавателем употребление чисел в разных языках программирования (int-целое число, float – дробное число (число с плавающей точкой, number-число, double-числа с плавающей точкой двойной точности).
2. Определите тип данных для чисел: 43; 43,4 в JavaScript. Результат выведите в консоль и покажите преподавателю. Для определения используйте команду type of, для вывода данных в консоль – команду console.log. Объясните преподавателю полученный результат типа данных.

Часть 2. Строка.

3. Дайте определение понятию строка. Проанализируйте вместе с преподавателем, из чего могут состоять строки и как они обозначаются в языках программирования (Python, JavaScript); как тип данных «строка» записывается в коде алгоритма? Какие манипуляции можно проводить с этим типом данных?
4. Пропишите в коде любое слово на латинице таким образом, чтобы в поле консоль отобразился тип данных «строка». Покажите результат преподавателю.
5. Проделайте то же самое, только со словом на кириллице, результаты покажите преподавателю и вместе проанализируйте.
6. Поместите в строку выражение «It’s a cloudy day». Удовлетворяет ли вас полученных результат? Если нет, то почему? Проанализируйте результат с преподавателем.
7. Поместите данное выражение в строку еще раз, но уже используйте метод экранирования.
8. Определите индексы типа данных «строка». Как определить их позиции? Для выбора индекса как его обозначают? Как обозначить два индекса одновременно?
9. Покажите, что слово «BIBLIOTHEK» имеет тип данных «строка». Теперь выведите в консоль букву T при помощи индекса (для

вывода используйте команду .charAt). Проанализируйте с преподавателем использование данной команды.

10. Теперь выведите в отдельном коде буквы: I, B, E.

11. Обрежьте строку, используя метод .substring (подстрока) с индекса №4. Выведите результат в консоль и покажите результат преподавателю, обсудите специфику усечения при помощи данного метода.

12. Применив метод «substring», используйте индексы 2, 5, 7, 3. Покажите результат преподавателю.

13. Теперь попробуйте вывести данные, используя усечения индексов с 3 до 5. Покажите результат преподавателю и обсудите его.

14. Методом «substring» поработайте с усечениями 4-6, 1-5, 2-8. Покажите результаты преподавателю и проанализируйте использование данного метода.

15. Обрежьте сроку, используя метод .slice (обрезать)с индекса №4. Покажите результат преподавателю, обсудите специфику усечения при помощи данного метода.

16. Используя данный метод, представьте индекс в отрицательном значении, выведите результат в консоль, покажите преподавателю, обсудите специфику отрицательного представления индекса.

17. Применив метод «slice» используйте отрицательные индексы -6, -7, -3. Результаты выведите в консоль и покажите преподавателю.

18. Обрежьте строку, используя метод .substrc индекса 4. Покажите результат преподавателю, обсудите специфику усечения при помощи данного метода.

19. Выведите в консоль данные, используя усечения индексов с 3 до 5 методом «substr». Покажите результат преподавателю. Обсудите с преподавателем специфику использования данного метода.

20. Методом «substr» поработайте с усечениями 2-3, 5-2, 0-6. Покажите результаты преподавателю.

Часть 3. Булевый тип (логический).

21. Когда применяется булевый тип данных? Как данный тип обозначаются в языках программирования?

22. Пропишите булевые операторы в коде, определяя их тип данных. Результат выведите в консоль и покажите преподавателю.

23. Составьте логический пример, с положительным булевым результатом. Выведите в консоль, результат покажите преподавателю. Обсудите с преподавателем использование знака равенства в языках программирования.

24. Составьте логический пример с отрицательным булевым результатом. Выведите в консоль, результат покажите преподавателю.

25. Зная основные типы данных, преобразуйте один тип данных в другой: результат выведите в консоль и покажите преподавателю.

Содержание отчета по работе: сохраненный на персональном компьютере файл.

Литература:

1. ГОСТ 19.701–90 (ИСО 5807–85) Единая система программной документации. Схемы алгоритмов, программ, данных и систем: обозначения условные и правила дополнения. – М.: Стандартинформ, 2010. – 158 с.

2. Базовые типы данных [Электронный ресурс]. – Режим доступа: https://works.doklad.ru/view/RePiip1RHTI.html. – Дата доступа: 03.11.2019.

3. Типы данных JavaScript и структуры данных [Электронный ресурс]. – Режим доступа: https://developer.mozilla.org/ru/docs/Web/JavaScript/Data_structures. – Дата доступа: 03.11.2019.

ЛАБОРАТОРНАЯ РАБОТА №3
па теме **«Основные структуры данных»**, 4 часа

Цель: познакомиться с основными структурами данных в языках программирования: переменная, массив; изучить специфику использования структур данных в языке программирования JavaScript.

Алгоритм выполнения лабораторной работы:

1. Проанализируйте вместе с преподавателем особенности кода переменной, а также варианты использования переменной и массива.
2. В JavaScript объявите переменную, используя ключевое слово var, затем дайте имя вашей переменной (используйте тип данных «строка») и через пробел пропишите его.
3. Запишите значение переменной, используя оператор присваивания «=» (знак равенства).
4. Выведите результат в консоль и покажите преподавателю.
5. Обсудите с преподавателем разницу между переменной и значением переменной, а также правила именования переменной.
6. Создайте свои примеры переменной, используя другие типы данных. Результат покажите преподавателю.
7. Давайте создадим переменную с вызовом диалогового окна (вопрос: в каких случаях в АБИС мы можем видеть примеры использования диалоговых окон?).
8. Объявите переменную и дайте ей имя Name, присвойте значение переменной, используя тип данных «строка» (значение переменной отобразит информацию диалогового окна).
9. Пропишите после оператора присваивания команду вызова диалогового окна «prompt».
10. Пропишите вывод команды в консоль, значением для консоли служит имя переменной.
11. Отправьте алгоритм на выполнение, в появившемся диалоговом окне пропишите ответ на вопрос диалогового окна, нажмите «Ок», и покажите результат преподавателю. Специфику данного алгоритма и результат проанализируйте вместе с преподавателем.
12. Создайте свои примеры алгоритма с вызовом диалогового окна, используя разные типы данных. Результаты обсудите с преподавателем.
13. Обсудите с преподавателем использование в АБИС массивов данных, подходящие для массивов типы данных, простые и структурированные элементы массива, многоуровневые массивы, индексы элементов массива, запись элементов массива и самого массива.

14. Создайте массив, дайте ему имя и пропишите несколько элементов массива, используя тип данных «строка».

Пример: var List = [‘cheese’, ‘brot’, ‘butter’, ‘wurst’, ‘apple’];

15. Выведите в консоль весь массив (помним, что значение массива мы можем увидеть только через использование его имени). Покажите результат преподавателю.

16. Выведите первый элемент массива, покажите результат преподавателю и проанализируйте вместе специфику и использование кода.

17. Выведите каждый элемент из массива, используя соответствующие индексы.

18. Создайте еще один массив, под именем Drinks и включите в элементы массива [‘milc’, ‘cola', 'tea', ‘saft'], выведите результат в консоль и покажите преподавателю.

19. Объедините два массива, поместив имя массива Drinks на второе место элемента массива List.

20. Обратите внимание на расположение массивов одного над другим.

21. Выведите в консоль весь массив Drinks (используйте индекс расположения элементов массива). Результат проанализируйте с преподавателем.

22. Выведите в консоль элемент внутреннего массива (над-массива) в консоль, результат покажите преподавателю. Проанализируйте использование индексов массивов.

23. Раскройте в консоли все элементы внутреннего массива поочередно, результат покажите преподавателю.

24. Выведите одновременно несколько элементов массивов внутреннего и внешнего, результат покажите преподавателю и проанализируйте специфику использования одновременно двух массивов.

25. Создайте по образцу свой пример многоуровневого массива, используя разные типы данных.

Содержание отчета по работе: сохраненный на персональном компьютере файл.

Литература:

1. Основные структуры данных [Электронный ресурс]. – Режим доступа: http://khpi-iip.mipk.kharkiv.edu/library/datastr/book_sod/guap/index1.htm. – Дата доступа: 03.12.2019.

2. Типы данных JavaScript и структуры данных [Электронный ресурс]. – Режим доступа: https://developer.mozilla.org/ru/docs/Web/JavaScript/Data_structures. – Дата доступа: 03.12.2019.

ЛАБОРАТОРНАЯ РАБОТА №4
па теме **«Основные структуры данных: функции»**, 6 часов

Цель: изучить специфику использования функций в языке программирования JavaScript.

Алгоритм выполнения лабораторной работы:

1. Вспомните из курса лекций, что такое функция. Проанализируйте, вместе с преподавателем, для чего нужны функции в коде, и как они оформляются; какие переменные можно передавать в параметры функции; что в функции принято называть Аргументом и т.д.
2. Теперь напишем пример функции. Чтобы создать новую функцию нужно записать наш алгоритм в определенной «обертке» и дате ему имя, а потом вызвать с помощью команды этого имени.
3. Определите новую функцию, начав с ключевого слова объявления функции function.
4. Далее, через пробел дайте название функции (например, greeting (функция приветствия)).
5. В скобках после названия функции передаются параметры (аргументы) функции.
6. После прописываем тело функции с выводом в консоль (например, 'Hallo! It’s a Javascript Function!’) (не забываем про экранирование элементов).
7. Оборачиваем тело функции в фигурные скобки.
8. Командой имени функции вызываем ее и смотрим результат в консоли.
9. Удалите предыдущий код и пропишите свой пример функции. Результат выведите в консоль и покажите преподавателю.
10. Теперь создадим еще одну функцию, но уже со значениями, с которыми будущая функция будет работать.
11. Создадим функцию и назовем ее height и передадим ей параметры (m, cm). Обратите внимание на то, что расположения параметров и отступы будут точно также отражены и в значении функции.
12. Теперь пропишите тело функции через объявление переменной и имени (например, total) и присвойте ей значение через простую формулу перевода метров в сантиметры (100*m)+cm.
13. Пропишите вывод переменной в консоль через имя переменной +cm.
14. Закройте тело функции и вызовите название функции, передав в скобках значения параметров.
15. Результат выведите в консоль и проанализируйте с преподавателем использование функции со значением параметров.

16. Вместо параметров мы можем передать любое числовое значение, формула пересчитает и выведет правильный результат. Передайте несколько вариантов значений для проверки работы функции.

17. Обратим внимание, что результат кода получился двухуровневый, так как использовались в теле функции несколько значений. Вам нужно привести результат к одному уровню посредством конкатенации строк.

18. Разберите вместе с преподавателем что такое «конкатенация», ее оформление в коде и проделайте ее в своем алгоритме.

19. Самостоятельно оформите функцию веса, или количества по аналогии с функцией height. Не забудьте про конкатенацию. Результат покажите преподавателю.

20. Проанализируйте с преподавателем использование функций возврата значения в код программы. Теперь пропишем такую функцию.

21. Создайте функцию calc и дайте ей два параметра (a, b).

22. В теле функции объявите переменную sum и присвойте ей значение формулой сложения параметров.

23. Пропишите возврат значения функции через команду return (помним, что значение функции выводим посредством обращения кода к имени функции.

24. Закройте тело функции и выведите в консоль значение через имя функции.

25. Подставьте любые значения в параметры функции (например, 4, 5). И отправьте функцию на выполнение.

26. Подставьте еще несколько числовых вариантов для того, чтобы убедиться, что алгоритм правильно работает. Результат работы функции обсудите с преподавателем.

27. Далее самостоятельно напишите функции возврата на умножение, деление и вычитание. Протестируйте их, задав несколько параметральных значений. Результаты выполнения такого типа функций в коде обсудите с преподавателем.

Содержание отчета по работе: сохраненный на персональном компьютере файл.

Литература:

1. Основы JavaScript: функции [Электронный ресурс] // Язык программирования JavaScript. – Режим доступа: https://learn.javascript.ru/function-basics. – Дата доступа: 04.11.2019.

2. JavaScript функции [Электронный ресурс] // HTML, CSS, JavaScript, jQuery, HDOM, AJAX. – Режим доступа: http://www.wisdomweb.ru/JS/func.php. – Дата доступа: 04.11.2019.

ЛАБОРАТОРНАЯ РАБОТА №5
па теме **«Управление процессом выполнения программы: условия»**, 6 часов

Цель: научиться работать с условиями логического выбора; строить алгоритмические цепочки посредством единственного, двойного и множественного выборов.

Алгоритм выполнения лабораторной работы:

1. Обсудите с преподавателем условия управления процессами выполнения программой; вспомните виды алгоритмов и проанализируйте, какой из них используется в вышеназванных процессах; как логически будут определены условия алгоритмов и т.д.
2. Проанализируйте вместе с преподавателем употребление утверждения условия с использованием единственного выбора.
3. Создайте алгоритм с условием единственного выбора. Для этого создайте переменную и назовите ее (например, year).
4. Объявите оператор единственного выбора if и в скобках пропишите условия (например, год не равен 2019-му – year !=2019).
5. В фигурных скобках пропишите результат этой ветви алгоритма, используя тип данных строка (например, ‘Нет, сейчас другой год’), не забудьте команду вывода в консоль, чтобы данная ветвь алгоритма сработала при условии ее выбора.
6. За телом данной ветви пропишите код последующей операции, при ином выборе (например, ‘теперь 2019, скоро 2020’), не забудьте перед ним прописать команду вывода в консоль, чтобы сработала и другая ветвь.
7. Далее подставьте в условие значение не равное 2019 (посмотрите результат) и равное 2019 (посмотрите результат).
8. Объясните преподавателю, как при заданных условиях с использованием единственного выбора работают ветви алгоритма.
9. Используя условие единственного выбора, составьте свой пример алгоритма. Результат покажите преподавателю.
10. Далее проанализируйте вместе с преподавателем употребление условия с использованием двойного выбора.
11. Постройте алгоритм с условием двойного выбора. Для этого объявите ту же переменную, как и в предыдущем алгоритме и пропишите те же условия. Только в данном случае мы будет использовать оператор двойного выбора if/else.

12. Перед первой ветвью алгоритма поставьте первую часть оператора двойного выбора «if», проверьте условия, не забудьте закрыть тело ветви фигурной скобкой.

13. Перед второй ветвью алгоритма поставьте вторую часть оператора множественного выбора «else» и в теле этой ветви пропишите условие вывода, используя тип данных «строка» (console.log (‘Все верно, еще 2019’)), не забудьте поместить ветвь в фигурные скобки.

14. За телом второй ветви пропишите код последующей операции, при ином выборе (например, ‘скоро 2020’), не забудьте перед ним прописать команду вывода в консоль, чтобы сработал код, который показывает работы за ветвями условия двойного выбора.

15. Далее подставьте в условие значение не равное 2019 (посмотрите результат) и равное 2019 (посмотрите результат).

16. Объясните преподавателю, как при заданных условиях с использованием двойного выбора работают ветви алгоритма.

17. Используя условие двойного выбора, составьте свой пример алгоритма. Результат покажите преподавателю.

18. Далее проанализируйте вместе с преподавателем употребление условия с использованием множественного выбора.

19. Постройте алгоритм с условием множественного выбора. Для этого объявите переменную после знака присвоения значения переменной условие не пока не прописывайте.

20. Далее, используя оператор множественного выбора else if, постройте первую ветвь условия (используйте первую часть оператора множественного выбора if и задайте ему условие, где год равен 2019.

21. В теле ветви пропишите результат выполнения условия типом данных строка (например, ‘Все верно, еще 2019!’).

22. Создайте новую ветвь, используя оператор множественного выбора else if, и пропишите условие, где год будет равен 2020-му, а в теле ветви пропишите результат выполнения условия типом данных строка (например, ‘Еще нет, но 2020 уже скоро!’).

23. Создайте еще одну ветвь, используя оператор множественного выбора else if, и пропишите условие, где год будет меньше либо равен 2018.

24. В теле ветви пропишите результат выполнения условия типом данных строка (например, ‘Это уже прошлое!’).

25. Создайте еще одно ветвь, используя оператор множественного выбора else if, и пропишите условие, где год будет больше либо равен 2021.

26. В теле ветви пропишите результат выполнения условия типом данных строка (например, ‘Это время пока не наступило!’).

27. Запустите алгоритм в действие, присвоив значения в объявленную переменную: 2019, 2020, 1998, 1204, 862, 2034, 2998 и т.д.

28. Проанализируйте с преподавателем работу каждой ветви алгоритма при заданном условии.

29. Теперь вместо ограничения, которое мы прописали в последней ветви, задайте значение алгоритма «остальные варианты».

30. Для этого вместо оператора множественного выбора else if, задайте часть оператора, отвечающую за иные варианты кода, не прописанные ограничением «else».

31. Условие в этом случае прописывать не нужно, а следует сразу перейти к телу ветви. Оставьте ее прежней, как в п. 26.

32. Проанализируйте с преподавателем работу кода данной ветви. Обоснуйте возможности использования двух последних вариантов завершения условия множественного выбора. И примеры, в каких случаях удобнее (логичнее) использовать тот либо иной вариант.

Содержание отчета по работе: сохраненный на персональном компьютере файл.

Литература:

1. Условные операторы в JavaScript: конструкция IF-ELSE [Электронный ресурс] // Изучаем JavaScript. – Режим доступа: http://www.webpupil.ru/javascript_step_view.php?id=10. – Дата доступа: 13.11.2019.

2. Условные операторы [Электронный ресурс] // JavaScript.ru: учебник. – Режим доступа: https://learn.javascript.ru/ifelse10. – Дата доступа: 13.11.2019.

ЛАБОРАТОРНАЯ РАБОТА №6
па теме **«Управление процессом выполнения программы: циклы»**, 6 часов

Цель: овладеть технологией работы с циклами как инструментом управления выполнением программой.

Алгоритм выполнения лабораторной работы:

1. Обсудите с преподавателем, что представляет собой цикл в работе алгоритма, для чего он используется, из чего состоит. Какие есть типы циклов. Что такое итерация.
2. Безусловный цикл. Пропишите в коде пример безусловного цикла, он будет прорабатываться кодом n-ное количество раз, но программа JavaScript ограничит его выполнение.
3. Проанализируйте с преподавателем его специфику, как он влияет на процессор, и как из него можно выйти, если все-таки цикл запустился.
4. Цикл с предусловием. Обсудите с преподавателем использование циклов с предусловием, их специфику выполнения, какие необходимы условия для выполнения данного типа цикла.
5. Создайте цикл с предусловием while. Для этого объявите переменную и назовите ее digit, пропишите ей значение =1.
6. Далее напишите сам цикл с while, и пропишите условие: пока переменная меньше либо равна 100, то (прописывается в теле цикла) выполняем следующий код: вывод в консоль результат переменной ‘ok’.
7. Далее, пропишите шаг переменной (например, на единицу больше) и закройте тело цикла. За телом в консоль выведите код остальных вариантов переменной: console.log (digit+ ‘more than 100!’).
8. Проверьте работу цикла. Создайте свой пример цикла с предусловием while. Результат покажите преподавателю.
9. Цикл с постусловием. Обсудите с преподавателем использование циклов с постусловием, их специфику выполнения, какие необходимы условия для выполнения данного типа цикла.
10. Создайте цикл с постусловием do…. Для этого создайте переменную и назовите ее digit, но значение пока не прописывайте.
11. Пропишите первую часть оператора цикла do и в фигурных скобках пропишите тело цикла со следующим значением: выводить результат работы цикла ‘ok!’ с шагом в +1. После тела while цикла добавьте условие, используя вторую часть оператора цикла while: в том случае, если переменная будет меньше либо равна 10; после завершения цикла с

постусловием, когда результат больше 10, пропишите код вывода: (digit+ ‘more than 100!’).

12. Теперь в условие цикла пропишите поочередно значения 2; 5; 11 и отправьте на выполнение цикл. Результат обсудите с преподавателем. В каких случаях используется цикл с постусловием?

13. Создайте свой пример цикла с постусловием.

14. Цикл со счетчиком. Обсудите с преподавателем использование циклов со счетчиком, их специфику выполнения, какие необходимы условия для выполнения данного типа цикла. А также порядок выполнения данного цикла (инициализация, условие, блок кода, обновление, переход).

15. Создайте цикл со счетчиком for. Для этого создайте инициализирующую переменную (в языках программирования ее принято называть i) она и выполняет роль счетчика.

16. Далее пропишите условия for (i=0, i<=0) и обновление, в условии увеличивающееся на 1. (++)

for (i=0, <=0; i++{

17. Теперь создайте переменную arr со значением var arr= и цифровой массив [1,4,6, 3, 10].

18. Добавьте в инициализации еще одну переменную, назовите ее sum, значение прировняйте к нулю for (i=0, sum=0; i<arr.llenght; i++) {.

19. Далее, добавьте значение массива, взяв индекс данного элемента массива, и если получить сумму всех элементах массива, то от нуля до всей длинны массива командой arr.llenght.

20. Пропишите в коде сумму всех элементов массива не зависимо от его длинны.

21. var arr=[1,4,6, 3, 10];

```
for (i=0, sum=0; i<arr.llenght; i++){
sum+=arr[i];
};
console.log (sum);
```

22. Создайте свой пример цикла со счетчиком. Результат покажите преподавателю.

23. Цикл просмотра. Обсудите с преподавателем использование циклов просмотра, их специфику выполнения, какие необходимы условия для выполнения данного типа цикла.

24. Создайте цикл просмотра типа for in с условием: выполнить операцию x для всех элементов входящих во множество y. Обратите внимание, что цикл типа for in возвращает именно имеющиеся индексы, а не значения, а уже по индексам мы можем получить сами значения.

25. Цикл с выходом из середины. Обсудите с преподавателем использование циклов с выходом из середины, их специфику выполнения, какие необходимы условия для выполнения данного типа цикла. Проанализируйте специфику использования команд возврата: break, continuum.

26. На основе примера п.16-22, дополните код командой break, для выхода цикла с середины, прописав условие: если сумма массива равна 6, цикл выходит из середины, и продолжает код за телом цикла. Результат покажите преподавателю.

27. На этом же примере создайте цикл с выходом из середины командой continuum, добавив в условие, кроме суммы массива =6, исключить из условия 10. Отправьте цикл на выполнение. Результат покажите преподавателю. Обсудите выполнение двух циклов, их специфику и условия использования.

28. Создайте цикл с выходом из середины break, с любым другим условием работы массива кроме суммы, с таким же условием создайте цикл с выходом из середины командой continuum. Результат покажите преподавателю. Обсудите выполнение двух циклов, их специфику и условия использования.

Содержание отчета по работе: сохраненный на персональном компьютере файл.

Литература:

1. Циклы и итерации [Электронный ресурс] // JavaScript.ru: учебник. – Режим доступа: https://www.google.by/url?esrc=s&q=&rct=j&sa=U&url=https://developer.mozilla.org/ru/docs/Web/JavaScript/Guide/%25D0%25A6%25D0%25B8%25D0%25BA%25D0%25BB%25D1%258B_%25D0%25B8_%25D0%25B8%25D1%2582%25D0%25B5%25D1%2580%25D0%25B0%25D1%2586%25D0%25B8%25D0%25B8&ved=2ahUKEwiL7oqst6jmAhURDewKHUYjC8AQFjABegQIBBAB&usg=AOvVaw2htzXL8wVTX43PwHVQAev_. – Дата доступа: 23.11.2019.

2. Циклы в JavaScript [Электронный ресурс]. – Режим доступа: https://www.google.by/url?esrc=s&q=&rct=j&sa=U&url=https://monsterlessons.com/project/lessons/cikly-v-javascript&ved=2ahUKEwiL7oqst6jmAhURDewKHUYjC8AQFjAGegQIAxAB&usg=AOvVaw2me9ukObTmUzoVMeldRn37. – Дата доступа: 23.11.2019.

Методические указания к семинарским занятиям

Семинарские занятия являются результатом самостоятельной работы по изучению основной и дополнительной литературы по предложенной ниже тематике. Семинарские занятия способствуют более глубокому осмыслению и усвоению самых сложных и значимых тем ученой дисциплины.

На семинарских занятиях студенты должны показать достаточно глубокое знание материала темы, умения анализировать этот материал, самостоятельно рассуждать и делать теоретические и практические выводы по определенной теме.

В подготовку к семинарскому занятию входит изучение рекомендуемой литературы, конспектирование, составление развернутых ответов по каждому вопросу. Необходимо прочитать литературу, сделать выписки и подготовить краткий или развернутый (по желанию студента) письменный ответ на каждый вопрос семинара. Этими записями студент может пользоваться при ответах на семинарских занятиях.

Существуют следующие требования к выступлению студента: хорошее владение изученным материалом; умение связать его с соответствующим направлением деятельности библиотек; логичность и последовательность в ответе; выделение наиболее важных положений и умение сделать выводы по каждому вопросу семинара. Дополнительные баллы набирают студенты, которые дополняют выступления, высказывают критические и оценочные суждения, ведут дискуссию, готовят небольшие презентации по тематике семинарских занятий.

Тематика семинарских занятий

№ п/п	Тема семинарского занятия	Кол-во ауд. часов
1	Теоретические основы алгоритмизации	2
2	Теоретические вопросы программирования	2
	Всего	**4**

СЕМИНАР №1

по теме **«Теоретические основы алгоритмизации»,** 2 часа

Вопросы:

1. Дайте определение понятия «алгоритм» и раскройте свойства алгоритма.
2. Каковы изобразительные средства для описания алгоритмов?
3. Раскройте особенности графического изображения алгоритмов и представьте их схемы.
4. Раскройте виды алгоритмов: алгоритмы линейной структуры.
5. Раскройте виды алгоритмов: алгоритмы разветвляющейся структуры.
6. Раскройте виды алгоритмов: алгоритмы циклической структуры.

Литература:

1. Голицына, О. Л. Основы алгоритмизации и программирования : учеб. пособие для СПО / О. Л. Голицына, И. И. Попов. – 3-е изд., испр. и доп. – М. : Форум, 2008. – 432 с.
2. Жданова, Т. А. Основы алгоритмизации и программирования : учеб. пособие / Т. А. Жданова, Ю. С. Бузыкова. – Хабаровск : Изд-во Тихоокеан. гос. ун-та, 2011. – 56 с. – Режим доступа: http://window.edu.ru/resource/402/77402. – Дата доступа: 06.11.2019.
3. Алгоритмизация и программирование : учеб. пособие для студентов учреждений среднего профессионального образования, обучающихся по группе специальностей "Информатика и вычислительная техника" / С. А. Канцедал. – М. : Форум : Инфра-М, 2013. – 351 с.
4. Семакин, И. Г. Основы алгоритмизации и программирования : учебник для среднего профессионального образования / И. Г. Семакин, А. П. Шестаков. – М. : Академия, 2010. – 232 с.

СЕМИНАР №2

по теме **«Теоретические вопросы программирования»,** 2 часа

Вопросы:

1. Дайте определение понятия «программы», раскройте назначение прикладных и системных программ.
2. Каковы основные принципы программирования: классификация технологий программирования?
3. Каковы алфавиты и лексемы современных языков программирования?
4. Раскройте классификацию стандартных операций современных языков программирования.
5. Определите виды разделителей в современных языках программирования.

Литература:

1. Голицына, О. Л. Основы алгоритмизации и программирования : учеб. пособие для СПО / О. Л. Голицына, И. И. Попов. – 3-е изд., испр. и доп. – М. : Форум, 2008. – 432 с.
2. Жданова, Т. А. Основы алгоритмизации и программирования : учеб. пособие / Т.А. Жданова, Ю.С. Бузыкова. – Хабаровск : Изд-во Тихоокеан. гос. ун-та, 2011. – 56 с. – Режим доступа: http://window.edu.ru/resource/402/77402. – Дата доступа: 06.11.2019.
3. Алгоритмизация и программирование : учеб. пособие для студентов учреждений среднего профессионального образования, обучающихся по группе специальностей "Информатика и вычислительная техника" / С. А. Канцедал. – М. : Форум : Инфра-М, 2013. – 351 с.
4. Семакин, И. Г. Основы алгоритмизации и программирования : учебник для среднего профессионального образования / И. Г. Семакин, А. П. Шестаков. – М. : Академия, 2010. – 232 с.

РАЗДЕЛ КОНТРОЛЯ ЗНАНИЙ

Методические указания к контролируемой самостоятельной работе

В целях повышения эффективности усвоения учебного материала по дисциплине и формирования профессиональных компетенций учебным планом специальности предусмотрена самостоятельная работа студентов, которая направлена на активизацию учебно-познавательной деятельности студентов; формирование у них умений и навыков самостоятельного приобретения и обобщения знаний; формирование умений и навыков самостоятельного применения знаний на практике; саморазвитие и самосовершенствование. Самостоятельная работа помогает организовать равномерную деятельность студентов по изучению предмета, более глубоко и основательно усвоить его, достичь своевременного выполнения заданий, следить за индивидуальным рейтингом.

Самостоятельная работа студентов, осваивающих образовательные программы I ступени высшего образования, рассматривается как целенаправленная, внутренне мотивированная, структурированная и корректируемая самими субъектами образовательного процесса деятельность по самостоятельному изучение отдельных тем (блоков) учебной дисциплины (выполнение письменных заданий, подготовка сообщения по выбранной теме, составление мини-глоссария, тестовых заданий, написание эссе). Самостоятельная работа организуется студентом с учетом своих психологических особенностей и личностной заинтересованности.

Управляемая самостоятельная работа студента осуществляется непосредственно под руководством преподавателя и направлена на изучение студентами теоретического, практико-ориентированного материала, что позволяет последним выработать личностную позицию по изучаемой учебной дисциплине, получить представления о возможности применение полученной информации в своей профессиональной деятельности.

Результаты самостоятельной работы студентов могут обсуждаться в рамках семинарских занятий (как отдельный вопрос), так и в рамках итоговой аттестации.

Для самостоятельного изучении разделов и тем по данной учебной дисциплине студенты могут использовать труды профессорско-преподавательского состава университета, размещенные в Репозитории Белорусского государственного университета культуры и искусств (http://repository.buk.by), в БД собственной генерации библиотеки БГУКИ – «Труды преподавателей, сотрудников, аспирантов, магистрантов и студентов БГУКИ»; пользоваться полнотекстовыми электронными ресурсами, доступ к

которым открыт для субъектов и объектов образовательного процесса университета.

Управляемая самостоятельная работа имеет практико-ориентированный характер и направлена на закрепление навыков программирования.

Тематика и описание контролируемой самостоятельной работы

№ п/п	Тема самостоятельной работы	Кол-во часов
1	Арифметические, логические операторы, операторы сравнения в JavaScript	6
2	Функции. Операторы цикла. Объекты Math, Number	4
3	Встроенные объекты JavaScript	4
	Всего	**14**

Тема 1. Арифметические, логические операторы, операторы сравнения в JavaScript, 6 часов

Цель: изучить возможности ввода и вывода данных; особенности использования арифметических, логических операторов, операторов сравнения, операторов условного перехода.

Задания для СРС:

Задание 1. Опробовать работу с модальными окнами **alert**, **prompt**, **confirm**.

Разработать свой диалог с использованием этих команд. Пример диалога:

- вывести приветствие в окне alert “Вас приветствует кафедра ИРК”;
- затем в окне prompt вывести сообщение «Введите имя», а в окне alert вывести сообщение «Добро пожаловать на занятия, *имя*»;
- в окне confirm вывести сообщение "Хотите стать программистом?" с альтернативными ответами в окнах alert, если TRUE, то "Учите JavaScript!", если FALSE то "Упускаете время!".

Задание 2. Использование оператора + с числовыми и строковыми переменными.

1) сложить два любых числа (например, 10 и 5). Результат вывести на страницу;
2) сложить две строки (например, “10” и “5”). Результат вывести на страницу;
3) сложить число и строку (например, 22 и “5”). Результат вывести на страницу;
4) сложить строку и число (например, “22” и 5). Результат вывести на страницу;

5) вывод – *Результатом сложения строки и числа всегда будет …* – вывести в окно.

Задание 3. Найти значение 2-х любых арифметических выражений, записанных на JavaScript с использованием арифметических операций (например, 35y-25x)/5+232 и 8*y/x+5*x/y – 43)*6). Значения *x и y* задать произвольно, но подобрать такие, чтоб значение одного выражения было намного больше второго. Найти остаток от деления значения одного выражения на значение другого. Вывести результаты на страницу, а также в отдельное окно.

Задание 4. Продемонстрировать использование операторов сравнения (==, !=, >, >=, <, <=) и логических операторов (!, &&, ||).

Введите любое число. Если оно меньше 20 или больше 40 и не равно 15 и делится без остатка на 5, то вывести сообщение «Правильное значение», иначе «Не правильное значение».

Придумайте свой пример с логическими операторами.

Задание 5. Конструкция if…else if…else.

Ввести 2 числа с клавиатуры. Сравнить их и вывести сообщение на страницу, например, "*А больше Б*", используя конструкцию if…else if…else.

Написать свой пример с использованием альтернативного синтаксиса конструкции if...else (условие)? команды1:команды2

Задание 6. Определить, какой сегодня день недели (использовать конструкцию switch case).

Задание 7*. Придумать пример, демонстрирующий обработку исключений с использованием конструкции try … catch.

Конечный результат. Задание предоставляются преподавателю в распечатанном виде в предварительно оговоренные сроки.

Форма контроля СРС. Выступление на семинаре. Дополнительный вопрос на экзамене.

Литература:

1. Захаркина, В. В. JavaScript. Основы клиентского программирования : учеб. пособие / В. В. Захаркина. – СПб. : Ф-т филологии

и искусств СПбГУ, 2007. – 73 с. – Режим доступа: http://window.edu.ru/resource/394/57394. – Дата доступа: 06.11.2019.

2. Зудилова, Т. В. Web-программирование: JavaScript : учеб. пособие / Т. В. Зудилова, М. Л. Буркова. – СПб. : НИУ ИТМО, 2012. – 68 с. – Режим доступа: http://window.edu.ru/resource/612/76612. – Дата доступа: 06.11.2019.

3. Основы программирования : учеб. пособие / В. В. Борисенко ; Интернет-университет информационных технологий, [МГУ им. М. В. Ломоносова]. – М. : ИНТУИТ.ру, 2005. – 314 с.

4. Сафронов, И. К. JavaScript в задачах и примерах / И. К Сафронов. – СПб. : БХВ-Петербург, 2008. – 401 с.

Тема 2. Функции. Операторы цикла. Объекты Math, Number, 4 часа

Цель: научиться создавать пользовательские функции на JavaScript, использовать в программах операторы цикла; изучить особенности использования встроенных объектов **Math** и **Number**.

Задания для СРС:

Задание 1. Вывести таблицу умножения, **a** и **b** ввести с клавиатуры. Использовать оператор цикла **for.**

Таблица 1.

1	2	…	b
2	4	…	2*b
…			
a	a*2	…	a*b

Применить к ячейкам таблицы свойства форматирования.

Задание 2. Найти площадь круга и длину окружности, радиус меняется от **a** до **b** с шагом 0,3. Результаты округлить и вывести в таблице. Использовать оператор цикла **do-while.**

Таблица 2.

Радиус	Площадь круга	Длина окружности
a		
a+0,3		
…		
b		

Применить к ячейкам таблицы свойства форматирования.

Задание 3. Создать 2 объекта-числа (Number) – дробное и целое. Применить к каждому из них методы: **toExponentia**l, **toFixed**, **toPrecision**, **toString**. Результаты проанализировать, представить в таблице в следующем виде:

Таблица 3.

Число	Метод	Результат	Описание метода
127.18	toString(16)	7f.2e147ae147b	Строковое представление числа в 16-ричной системе счисления

Конечный результат. Задание предоставляются преподавателю в распечатанном виде в предварительно оговоренные сроки.

Форма контроля СРС. Выступление на семинаре. Дополнительный вопрос на экзамене.

Литература:

1. Захаркина, В. В. JavaScript. Основы клиентского программирования : учеб. пособие / В. В. Захаркина. – СПб. : Ф-т филологии и искусств СПбГУ, 2007. – 73 с. – Режим доступа: http://window.edu.ru/resource/394/57394. – Дата доступа: 06.11.2019.

2. Зудилова, Т. В. Web-программирование: JavaScript : учеб. пособие / Т. В. Зудилова, М. Л. Буркова. – СПб. : НИУ ИТМО, 2012. – 68 с. – Режим доступа: http://window.edu.ru/resource/612/76612. – Дата доступа: 06.11.2019.

3. Основы программирования : учеб. пособие / В. В. Борисенко ; Интернет-университет информационных технологий, [МГУ им. М. В. Ломоносова]. – М. : ИНТУИТ.ру, 2005. – 314 с.

4. Сафронов, И. К. JavaScript в задачах и примерах / И. К Сафронов. – СПб. : БХВ-Петербург, 2008. – 401 с.

Тема 3. Встроенные объекты JavaScript, 4 часа

Цель: изучить встроенные объекты JavaScript Array, String, Date, их свойства и методы, научиться их правильно применять в своих скриптах.

Задания для СРС:

Задание 1. Сформировать массив (объект Array), элементами которого являются значения выражений. Для вычисления выражений использовать объект Math. Найти максимальный и минимальный элементы массива и их номера.

1. $6 \cdot \pi^2 + 3 \cdot e^8$
2. $2\cos(4) + \cos(12) + 8 \cdot e^3$
3. $3\sin(9) + \ln(5) + \sqrt{3}$
4. $2\tan(5) + 6 \cdot \pi + \sqrt{12}$

Задание 2. Задан массив с элементами, представляющими методы объектов Array и Math (pow, pop, push, shift, round, floor, sline, sort). Получить из него 2 массива, в один записать методы объекта Array, в другой – методы объекта Math.

Добавить в начало одного массива и в конец другого еще по одному методу соответствующих объектов.

Вывести исходный массив, полученные массивы и их длину (количество элементов).

Задание 3. Создать объект **String** – строку текста (свои Фамилия Имя Отчество), в которой присутствуют строчные и прописные буквы. Узнать ее длину.

Перевести все символы строки в верхний регистр, а затем в нижний. Соединить полученные строки. Заменить свои Фамилия Имя Отчество на ФИО.

Вывести исходную и полученные строки на страницу.

Задание 4. Использовать объект **Date**. Вывести на страницу таблицу с составляющими текущей даты и времени в виде:

Таблица 8.

Год	2017
Месяц	
День	

Час	
Минуты	
Секунды	

Применить к ячейкам таблицы свойства форматирования.

Конечный результат. Задание предоставляются преподавателю в распечатанном виде в предварительно оговоренные сроки.

Форма контроля СРС. Выступление на семинаре. Дополнительный вопрос на экзамене.

Литература:

1. Захаркина, В. В. JavaScript. Основы клиентского программирования : учеб. пособие / В. В. Захаркина. – СПб. : Ф-т филологии и искусств СПбГУ, 2007. – 73 с. – Режим доступа: http://window.edu.ru/resource/394/57394. – Дата доступа: 06.11.2019.

2. Зудилова, Т. В. Web-программирование: JavaScript : учеб. пособие / Т. В. Зудилова, М. Л. Буркова. – СПб. : НИУ ИТМО, 2012. – 68 с. – Режим доступа: http://window.edu.ru/resource/612/76612. – Дата доступа: 06.11.2019.

3. Основы программирования : учеб. пособие / В. В. Борисенко ; Интернет-университет информационных технологий, [МГУ им. М. В. Ломоносова]. – М. : ИНТУИТ.ру, 2005. – 314 с.

4. Сафронов, И. К. JavaScript в задачах и примерах / И. К Сафронов. – СПб. : БХВ-Петербург, 2008. – 401 с.

ПРИМЕРЫ ТЕСТОВ ДЛЯ РУБЕЖНОГО КОНТРОЛЯ ЗНАНИЙ

по учебной дисциплине «Алгоритмизация и основы программирования»

1. Алгоритм применительно к ПК – это _____
_____
_____
_____
_____
_____

2. Из нижеперечисленного списка отметьте основные виды алгоритмов
   a) Линейный алгоритм
   b) Структурный алгоритм
   c) Циклический алгоритм
   d) Схематический алгоритм
   e) Табличный алгоритм
   f) Алгоритм ветвления
   g) Комбинированный алгоритм

3. Из нижеперечисленного списка отметьте свойства алгоритмов:
   a) Понятность
   b) Дискретность
   c) Целеполагание
   d) Определенность
   e) Формальность
   f) Результативной
   g) Коммуникативность
   h) Массовость

4. Какие основные типы данных используются для построения алгоритмов?
   a) Целое число
   b) Строка
   c) Буквенный тип
   d) Булевый тип
   e) Дробный тип
   f) Табличный тип

5. Каковы основные характеристики алгоритмов?
    a) Временные характеристики
    b) Пространственные характеристики
    c) Объемные характеристики
    d) Текущие характеристики

6. Какой командой выводится построенный алгоритм в консоль?
    a) Console.log
    b) Loftblog

7. Экранируйте фразу “It’s cloudy day” без вывода в консоль.

_____
_____

8. Каковы способы описания алгоритмов?
    a) Цифровой способ
    b) Словесный способ
    c) Графический способ
    d) Псевдокоды
    e) Линейный способ
    f) Компьютерный способ
    g) Программный способ

9. Из нижеперечисленного списка отметьте принципы фон Неймановской архитектуры:
    a) Принцип программного управления
    b) Принцип формулировки задачи
    c) Принцип однородности памяти
    d) Принцип функциональности
    e) Принцип адресности

10. Язык программирования – это _____
_____
_____
_____
_____
_____

11. Какие языки программирования Вы знаете?

_____

_____

12. Что такое HTML?

a) Язык программирования
b) Язык гипертекстовой разметки
c) Язык для написания алгоритма

13. Выберите правильный ответ: какая комбинация тегов составляет «скелет» сайта?

a)

<html>	
	<head>
	<body>
	</body>
	</head>
</html>	

b)

<html>	
	<head>
	</head>
	<body>
	</body>
</html>	

c)

<html>	
	<body>
	<head>
	</head>
	</body>
</html>	

14. Какими тегами обозначается «видимая» часть сайта в HTML?

a) <html> </html>
b) <head> </head>
c) <body> </body>

15. Сколько уровней заголовков используется в HTML?

a) 8 c) 6
b) 4 d) 7

16.Какими тегами выделяются заголовки 1 уровня в HTML?

a) <h1></h1>
b) <title1></title1>
c) <target1></target1>

17.С каким из нижеперечисленных правил работает версия HTML5?

a) <! DTD HTML>
b) <!DOCTYPE HTML PUBLIC>
c) <!DOCTYPE HTML>

ПЕРЕЧЕНЬ ВОПРОСОВ К ЭКЗАМЕНУ

по учебной дисциплине «Алгоритмизация и основы программирования»

1. Понятие алгоритма.
2. Свойства алгоритма: дискретность.
3. Свойства алгоритма: массовость.
4. Свойства алгоритма: определенность.
5. Свойства алгоритма: результативность.
6. Принципы построения алгоритмов.
7. Запись алгоритма на естественном языке.
8. Графический способ записи алгоритма.
9. Основные алгоритмические конструкции: линейная конструкция.
10. Основные алгоритмические конструкции: разветвленная конструкция.
11. Основные алгоритмические конструкции: циклическая конструкция.
12. Основные приемы алгоритмизации.
13. Адресация данных в памяти ЭВМ.
14. Классификация структур данных.
15. Файловая система хранения данных.
16. Понятие базы данных.
17. Основные операции по работе с базами данных.
18. Этапы решения задач на ЭВМ.
19. Этапы выполнения программ на ЭВМ.
20. Процесс создания программы: постановка задачи (входная и выходная информация).
21. Процесс создания программы: алгоритмизация решения задачи.
22. Языки программирования.
23. Понятия языка программирования.
24. Сущность объектно-ориентированного подхода к программированию.
25. Основные принципы объектно-ориентированного программирования.
26. Пример веб-программирования.
27. Основная характеристика языка разметки HTML.
28. Общая характеристика и назначение JavaScript.
29. Алфавит, синтаксис и семантика JavaScript.
30. Основные типы данных и переменных: числа и операторы.
31. Основные типы данных и переменных: переменные.
32. Основные типы данных и переменных: строки.

33. Основные типы данных и переменных: булевые значения.
34. Создание массива данных.
35. Доступ к элементам массива данных.
36. Разные типы данных в одном массиве.
37. Создание объектов.
38. Условные конструкции: конструкция if.
39. Условные конструкции: конструкция if… else.
40. Циклы: цикл while.
41. Циклы: цикл for.
42. Элемент canvas.
43. Общая характеристика PHP.
44. Назначение и применение языка PHP.

ПЕРЕЧЕНЬ СРЕДСТВ ДИАГНОСТИКИ РЕЗУЛЬТАТОВ УЧЕБНОЙ ДЕЯТЕЛЬНОСТИ

Для промежуточной и итоговой диагностики уровня знаний, умений и навыков студентов, полученных в процессе изучения учебной дисциплины «Алгоритмизация и основы программирования», проводится зачет и экзамен. Аттестация студентов осуществляется с учетом их академической активности на лекционных, семинарских, практических и лабораторных занятиях, а также с учетом выполненных ими учебных заданий в рамках контролируемой самостоятельной работы.

Основными видами контроля, обеспечивающими высокую степень диагностики уровня знаний, умений и навыков студентов по учебной дисциплине, являются:

– корректирующий контроль: экспресс-опрос в устной или письменной форме, собеседование по пройденному материалу;

– констатирующий контроль: оценка выступлений студентов с докладами и сообщениями на семинарских занятиях, проверка письменных работ (реферат, опорный конспект) или мультимедийных презентаций;

– самоконтроль: осуществляется самим студентом в форме анализа уровня своей подготовки по сравнению с одногруппниками;

– итоговый контроль: промежуточная и итоговая аттестация – тестирование, зачет и экзамен.

Для оценки качества самостоятельной работы студентов осуществляется систематический контроль за ее выполнением путем проверки выполненных студентами заданий в установленные преподавателем сроки.

КРИТЕРИИ ОЦЕНКИ ЗНАНИЙ СТУДЕНТОВ

10 (десять) баллов – самостоятельное, свободное, последовательное раскрытие темы (вопроса), подкрепленное ссылками из нескольких источников. Широкое владение терминологией. Собственный, аргументированный взгляд на затронутые проблемы. Предоставление тезисов.

9 (девять) баллов – свободное изложение содержания темы (вопроса), основанное на привлечении не менее трех источников, комментарии и выводы. Последовательность и ясность изложения материала. Широкое владение терминологией. Предоставление тезисов.

8 (восемь) баллов – то же, что и выше установлено. Некоторая незавершенность аргументации при изложении, требующая уточнения теоретических позиций. Владение терминологией.

7 (семь) баллов – понимание сути темы (вопроса), грамотное, но недостаточно полное изложение содержания. Отсутствие собственных оценок. Использование терминологии.

6 (шесть) баллов – понимание сути темы (вопроса), изложение содержания неполное, недостаточно свободное, требующее дополнений. Отсутствие собственных оценок. Неточности в терминологии.

5 (пять) баллов – поверхностная проработка темы (вопроса), неумение последовательно выстроить устное сообщение, невладение терминологией.

4 (четыре) балла – поверхностная проработка темы (вопроса), наличие некоторых погрешностей при ответе, пробелы в раскрытии содержания, невладение терминологией.

3 и 2 (три и два) балла – отсутствие знаний по значительной части основного учебно-программного материала, невыполнение предусмотренных программой основных заданий.

1 (один) балл – нет ответа (отказ от ответа).

ВСПОМОГАТЕЛЬНЫЙ РАЗДЕЛ

СОДЕРЖАНИЕ УЧЕБНОГО МАТЕРИАЛА

Тема 1. Введение. Основы теории алгоритмов

Объект, предмет, методы и задачи дисциплины. Математические основы теории алгоритмов. Основные задачи алгоритмизации.

Понятие алгоритма. Свойства алгоритмов. Общие принципы построения алгоритмов. Способы описания алгоритмов: словесно-формульное описание, блок-схема (схема графических символов), алгоритмические языки, операторные схемы, псевдокод. ГОСТ 19.002-80, ГОСТ 19.003-80.

Порядок выполнения алгоритма. Логически правильные алгоритмы. Эффективные алгоритмы.

Основные виды алгоритмов: линейные, разветвленные и циклические алгоритмы. Итеративные циклы. Вложенные циклы. Рекурсивные алгоритмы.

Применение алгоритмов в библиотечно-информационной деятельности. Основные приемы алгоритмизации. Примеры разработки и описания алгоритмов. Алгоритмы сортировки данных.

Тема 2. Основы технологии программирования

Этапы решения задач на ЭВМ. Кодирование и хранение информации в ЭВМ. Принципы организации вычислений на ЭВМ: принцип программного управления ЭВМ, принцип условного перехода, принцип хранимой программы, принцип использования двоичной системы счисления, принцип иерархичности запоминающих устройств. Принципы структурного программировании. Понятие о системе машинных команд.

Этапы выполнения программы на ЭВМ. Исходный код программы. Объектный код программы. Исполняемый код программы. Понятие компилятора. Понятие интерпретатора. Кроссплатформенность программных средств. Понятие системы программирования. Интегрированная инструментальная среда для разработки программ.

Языки программирования. История развития языков программирования. Классификация языков программирования. Процедурное программирование. Объектно-ориентированное программирование. Единая система программной документации. ГОСТ 19.001-77.

Понятие языка программирования. Алфавит, синтаксис и семантика языков программирования. Идентификаторы, служебные слова, разделители, комментарии.

Структура программы на языке высокого уровня. Модульный подход к созданию программ. Стиль программирования. Основы программной инженерии.

Тема 3. Основы организации работы с данными при создании программ

Адресация данных в памяти ЭВМ. Размещение данных и программы в памяти ЭВМ. Оперативная память, внешняя память, кэш-память.

Классификация структур данных. Статические и динамические структуры с последовательным или прямым доступом. Простые и составные структуры данных. Ассоциативные массивы. Понятия переменной и указателя. Одномерные и двумерные массивы данных.

Файловая система хранения данных. Понятие файла и файловой системы. Основные операции по работе с файлами данных. Основные стандарты кодирования символьных данных. Входные и выходные данные программы. Организация интерфейса с пользователем программы: интерфейс командной строки, графический интерфейс.

Понятие базы данных (БД). Использование языков программирования при работе с БД. Основы моделирования данных: сетевая, иерархическая и реляционная модели данных. Преимущества и недостатки реляционной модели данных. Целостность данных. Данные и метаданные. Понятие записи данных. Индексация данных. Основные функции системы управления базой данных (СУБД). Обеспечение доступа к БД. Понятие драйвера СУБД. Обеспечение доступа к БД из программы. Понятие курсора данных

Основные операции по работе с БД. Основы языка SQL. Создание и выполнение запросов к БД. Основы СУБД MySQL. Создание и выполнение запросов к БД. Администрирование базы данных. Понятие курсора данных.

Тема 4. Основы объектно-ориентированного программирования

Сущность объектно-ориентированного подхода (ООП) к программированию. Элементы объектной модели: понятие абстрагирования, понятие инкапсуляции, иерархия объектов. Основные принципы объектно-ориентированного программирования. Базовые понятия ООП. Классы и методы. Объекты и их свойства, методы и события.

Класс как основной механизм абстрагирования. Структура и организация класса. Управление доступа к членам класса. Спецификация и

реализация класса. Конструктор класса. Разнообразие классов. Отношения классов: обобщение, ассоциация, зависимость, реализация. Наследование как основная форма отношения обобщения. Объекты классов и полиморфизм. Иерархия классов и модули. Множественное наследование и интерфейсы. Виртуальные классы.

Конструктор класса. Инициализация полей объектов. Поля данных объектов и формальные параметры методов. Виртуальные методы. Конструктор. Динамические объекты. Внутреннее представление объектов.

Тема 5. Основы Веб-программирования

Предмет Веб-программирования. Основы технологии клиент-сервер. Основные прикладные протоколы Интернет. Понятие Веб-сервера. Характеристика распространенных Веб-серверов. Понятие Веб-браузера. Характеристика распространенных Веб-браузеров. Схема взаимодействия Веб-сервера и Веб-браузера.

Языки разметки. Основные понятия HTML. Структура документа HTML. Понятие тега. Формирование и структуризация контента HTML. Структура элемента Form документа HTML.

Протокол HTTP. Структура HTTP-запроса. Передача параметров серверу. Запоминание состояния. Меры безопасности CGI (Common Gateway Interface - общий шлюзовой интерфейс). Переменные окружения CGI. CGI и базы данных.

Программирование на стороне клиента и сервера. Инструменты и технологии Веб-программирования. Динамический HTML. Основы XML. Основы HTML5.

Тема 6. Основы программирования на JavaScript

История создания JavaScript. Общая характеристика и назначение JavaScript. Возможности JavaScript Области применения JavaScript. Основные компоненты JavaScript. ECMAscript.

Объектная модель JavaScript. Объектная модель браузера. Основные библиотеки JavaScript. Процесс разработки программы на JavaScript. Размещение программы на JavaScript. Современная разметка для тега SCRIPT. JavaScript и информационная безопасность. Альтернативные браузерные технологии. Плагины и расширения для браузера.

Объекты и Свойства. Функции и Методы. Определение методов в JavaScript. Создание новых объектов. Использование встроенных объектов и функций. Основные объекты языка JavaScript. Свойства и методы объекта

Date. Свойства и методы объекта String. Свойства и методы объекта window. Свойства и методы объекта/ Свойства и методы объекта event. Свойства и методы объекта Document. Свойства и методы объекта history. Свойства и методы объекта location. Свойства и методы объекта navigator. Окна и динамически создаваемые документы. Управление событиями.

Алфавит, синтаксис и семантика языка JavaScript. Структура кода. Конструкции языка JavaScript. Основные типы данных. Оператор type of. Структуры данных. Имена. Описание переменных. Локальные и глобальные переменные. Константы. Пользовательский тип. Бинарные и унарные операторы. Операции: математические, отношений, логические. Приоритеты операций. Встроенные функции. Операторы присваивания. Операторы управления. Условные операторы. Операторы организации цикла.

Программирование в среде фреймворков: назначение, построение, использование. Технология разработки программ на JavaScript.

Стандартные процедуры и функции. Описание пользовательских функций и процедур. Вызов функций и процедур и передача параметров по ссылке и по значению. Алгоритмизация и программирование математических вычислений по формулам на языке JavaScript. Функциональное наследование JavaScript.

Выявление и обработка ошибок ввода. Алгоритмизация и программирование обработки текстов. Анализ и обработка ошибок ввода и времени выполнения. Алгоритмизация и программирование логических задач. Алгоритмизация и программирование операций с датами и временем.

Массивы. Создание и обработка массивов данных. Преобразование массивов. Операции на массивах.

Тема 7. Основы программирования на PHP

Общая характеристика PHP. Назначение и применение языка PHP современной практике работы библиотек.

Синтаксис и грамматика. Константы и выражения. Функции. Классы. Операторы. Регулярные выражения. Арифметические операции. Операторы строк и присваивания. Бинарные операторы. Логические операторы. Операторы сравнения.

Разделение инструкций. Типы переменных. Инициализация массивов. Манипуляции с массивом. Инициализация объектов.

Область переменной. Глобальные переменные. Статическая переменная. Изменяемые переменные. Переменные вне скрипта PHP. Переменные окружения. Изменение типа. Определение типов переменных. Преобразование строк.

Функция date. Функции работы с файлами. Блокирование файлов. Доступ к базам данных. Пример базы данных. Пример сценария. Проверка и фильтрация данных. Установка соединения и выбор базы. Выполнение запроса. Получение результатов запроса. Ввод новой информации. Освобождение ресурсов. Создание и удаление баз данных. Работа с HTML-формами.

Аутентификация. Базовая аутентификация. Управление сеансами Аутентификация средствами управления сеансом.

УЧЕБНО-МЕТОДИЧЕСКАЯ КАРТА УЧЕБНОЙ ДИСЦИПЛИНЫ

Номер темы	Название темы	Количество аудиторных часов				Количество часов УСР	Форма контроля знаний
		Лекции	Практические занятия	Семинарские занятия	Лабораторные занятия		
1	Введение	1					
2	Основы теории алгоритмов	1	2	2	4	2	Опрос
3	Основы технологии программирования	2	2			2	Тест
4	Основы организации работы с данными при создании программ	4	2		2	2	Тест
5	Основы объектно-ориентированного программирования	4	2	2	2	4	Опрос
6	Основы Веб-программирования	4			2	2	Котроль-ная работа
7	Основы программирования на JavaScript	8			10	2	Котроль-ная работа
8	Основы программирования на PHP	6			8	4	Котроль-ная работа
	ИТОГО:	**30**	**8**	**4**	**28**	**18**	

ОСНОВНАЯ ЛИТЕРАТУРА

1. Голицына, О. Л. Основы алгоритмизации и программирования : учеб. пособие для СПО / О. Л. Голицына, И. И. Попов. – 3-е изд., испр. и доп. – М. : Форум, 2008. – 432 с.

2. Единая система программной документации. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения : государственный стандарт СССР ГОСТ 19.701-90 (ИСО 5807-85) : взамен ГОСТ 19.002-80 : взамен ГОСТ 19.003-80 : введен 01.01.92 / Государственный комитет СССР по управлению качеством продукции и стандартам. – М. : Изд-во стандартов, 1990. – 26 с.

3. Жданова, Т. А. Основы алгоритмизации и программирования : учеб. пособие / Т.А. Жданова, Ю.С. Бузыкова. – Хабаровск : Изд-во Тихоокеан. гос. ун-та, 2011. – 56 с. – Режим доступа: http://window.edu.ru/resource/402/77402. – Дата доступа: 06.11.2019.

4. Захаркина, В. В. JavaScript. Основы клиентского программирования : учеб. пособие / В. В. Захаркина. – СПб. : Ф-т филологии и искусств СПбГУ, 2007. – 73 с. – Режим доступа: http://window.edu.ru/resource/394/57394. – Дата доступа: 06.11.2019.

5. Зудилова, Т. В. Web-программирование: JavaScript : учеб. пособие / Т. В. Зудилова, М. Л. Буркова. – СПб. : НИУ ИТМО, 2012. – 68 с. – Режим доступа: http://window.edu.ru/resource/612/76612. – Дата доступа: 06.11.2019.

6. Лесневский, А. С. Объектно-ориентированное программирование для начинающих / А. С. Лесневский. – М. : Бином. Лаборатория знаний, 2009. – 212 с.

7. Основы программирования : учеб. пособие / В. В. Борисенко ; Интернет-университет информационных технологий, [МГУ им. М. В. Ломоносова]. – М. : ИНТУИТ.ру, 2005. – 314 с.

8. Основы программирования на PHP : курс лекций : учеб. пособие : для высших учебных заведений по специальностям в области информационных технологий / под ред. Н. В. Савельева ; Интернет университет информационных технологий. – М. : ИНТУИТ.ру, 2005. – 260 с.

9. Основы программирования на PHP : пер. с англ. / Ларри Ульман. – М. : ДМК пресс, 2001. – 285 с.

10. Павловская, Т. А. Программирование на языке высокого уровня : учеб. пособие для вузов / Т. А. Павловская. – СПб. : Питер, 2007. – 432 с.

11. Пархимович, М. Н. Основы интернет-технологий : учеб. пособие / М. Н. Пархимович, А. А. Липницкий, В. А. Некрасова ; Северный (Арктический) федер. ун-т им. М. В. Ломоносова. – Архангельск : ИПЦ

САФУ, 2013. – 366 с. – Режим доступа: http://biblioclub.ru/index.php?page=book&id=436379. – Дата доступа: 06.11.2019.

12. Савельева, Н. В. Язык программирования PHP / Н. В. Савельева. – 2-е изд., испр. – М. : Нац. Откр. Ун-т «ИНТУИТ», 2016. – 330 с. – Режим доступа: http://biblioclub.ru/index.php?page=book&id=428975. – Дата доступа: 06.11.2019.

13. Сафронов, И. К. JavaScript в задачах и примерах / И. К Сафронов. – СПб. : БХВ-Петербург, 2008. – 401 с.

14. Стесик, О. Л. Основы объектно-ориентированного программирования : учеб. пособие / О. Л. Стесик. – СПб. : Ф-т филологии и искусств СПбГУ, 2007. – 76 с. – Режим доступа: http://window.edu.ru/resource/391/57391. – Дата доступа: 06.11.2019.

15. Строганов, А. С. Ваш первый сайт с использованием PHP-скриптов : учеб. пособие / А.С. Строганов. – 3-е изд.. испр. и доп. – М. : Диалог-МИФИ, 2015. – 288 с. – Режим доступа: http://biblioclub.ru/index.php?page=book&id=447998. – Дата доступа: 06.11.2019.

ДОПОЛНИТЕЛЬНАЯ ЛИТЕРАТУРА

1. Алгоритмизация и программирование : учеб. пособие для студентов учреждений среднего профессионального образования, обучающихся по группе специальностей "Информатика и вычислительная техника" / С. А. Канцедал. – М. : Форум : Инфра-М, 2013. – 351 с.

2. Громов, Ю. Ю. Основы Web-инжиниринга: разработка клиентских приложений : учеб. пособие / Ю. Ю. Громов, О. Г. Иванова, С. В. Данилкин ; Тамбов. гос. техн. ун-т. – Тамбов : ФГБОУ ВПО «ТГТУ», 2012. – 240 с. – Режим доступа: http://biblioclub.ru/index.php?page=book&id=277648. – Дата доступа: 06.11.2019.

3. Диков, А. В. Веб-технологии HTML и CSS : учеб. пособие / А. В. Диков. – 2-е изд. – М. : Директ-Медиа, 2012. – 78 с. – Режим доступа: http://biblioclub.ru/index.php?page=book&id=96968. – Дата доступа: 06.11.2019.

4. Иванова, Г. С. Объектно-ориентированное программирование: учебник для вузов / Г. С. Иванова, Т. Н. Ничушкина, Е. К. Пугачев. – 2-е изд., перераб. и доп. – М. : МГТУ им. Н. Э. Баумана, 2007. – 368 с.

5. Изучаем программирование на JavaScript : [руководство по программированию на JavaScript] / Э. Фримен, Э. Робсон ; [пер. с англ. Е. Матвеев]. – СПб. : Питер : Питер Пресс, 2016. – 637 с.

6. Информатика : учеб. пособие / Тамбов. гос. техн. ун-т ; сост. Е. А. Ракитина [и др.]. – Тамбов : ФГБОУ ВПО «ТГТУ», 2015. – 159 с. – Режим доступа: http://biblioclub.ru/index.php?page=book&id=445045. – Дата доступа: 06.11.2019.

7. Информатика и основы программирования : учеб. пособие по специальности "Менеджмент организации" / М. Ф. Меняев. – М. : Омега-Л, 2005. – 463 с.

8. Информационные Web-технологии / Ю. Громов, О.Г. Иванова, Н.Г. Шахов, В.Г. Однолько ; Тамбов. гос. техн. ун-т. – Тамбов : ФГБОУ ВПО «ТГТУ», 2014. – 96 с. – Режим доступа: http: //biblioclub.ru/index.php?page=book&id=277935. – Дата доступа: 06.11.2019.

9. Карпиленко, Е. В. Основы программирования : учебник / Е. В. Карпиленко. – Ростов-н/Д. : Феникс, 2007. – 318 с.

10. Князева, М. Д. Алгоритмика : от алгоритма к программе : учеб. пособие / М. Д. Князева. – М. : КУДИЦ-ОБРАЗ, 2006. – 192 с.

11. Кормен, Т. Х. Алгоритмы : построение и анализ / Т. Х. Кормен. – 2-е изд. – СПб. : Вильямс, 2005. – 211 с.

12. Лыткина, Е. А. Основы языка HTML : учеб. пособие / Е. А. Лыткина, А. Г. Глотова ; Северный (Арктический) фед. ун-т им. М. В.

Ломоносова. – Архангельск : САФУ, 2014. – 104 с. – Режим доступа: http://biblioclub.ru/index.php?page=book&id=436328. – Дата доступа: 02.12.2019.

13. Мейер, Б. Основы объектно-ориентированного программирования / Б. Мейер. – М. : Интернет-Университет Информационных Технологий, 2005. – 1437 с. – Режим доступа: http://biblioclub.ru/index.php?page=book&id=234166. – Дата доступа: 06.11.2019.

14. Пестунов, А. И. Основы программирования : учеб. пособие / А. И. Пестунов ; Новосибир. гос. ун-т экономики и управления "НИНХ". – Новосибирск : НГУЭУ, 2014. – 183 с.

15. Семакин, И. Г. Основы алгоритмизации и программирования : учебник для среднего профессионального образования / И. Г. Семакин, А. П. Шестаков. – М. : Академия, 2010. – 232 с.

16. Сычев, А. В. Перспективные технологии и языки веб-разработки / А. В. Сычев. – 2-е изд., испр. – М. : Нац. Откр. Ун-т «ИНТУИТ», 2016. – 494 с. – Режим доступа: http://biblioclub.ru/index.php?page=book&id=429078. – Дата доступа: 06.11.2019.