МИНОБРНАУКИ РОССИИ

Орский гуманитарно-технологический институт (филиал)
федерального государственного бюджетного образовательного учреждения
высшего образования «Оренбургский государственный университет»
(Орский гуманитарно-технологический институт (филиал) ОГУ)

Кафедра программного обеспечения

Методические указания
для обучающихся по освоению дисциплины
«Б.1.В.ДВ.7.2 Разработка и реализация сетевых протоколов»

Уровень высшего образования

БАКАЛАВРИАТ

Направление подготовки
09.03.03 Прикладная информатика
(код и наименование направления подготовки)

Прикладная информатика в экономике
(наименование направленности (профиля) образовательной программы)

Тип образовательной программы
Программа академического бакалавриата

Квалификация
Бакалавр

Форма обучения
Очная

Год начала реализации программы (набора)
2014, 2015, 2016

г. Орск 2017

Методические указания для обучающихся по освоению дисциплины «Б.1.В.ДВ.7.2 Разработка и реализация сетевых протоколов» предназначены для обучающихся очной формы обучения направления подготовки 09.03.03 Прикладная информатика, профиля «Прикладная информатика в экономике»

Составитель _____ В.Н. Муллабаев

Методические указания рассмотрены и одобрены на заседании кафедры программного обеспечения, протокол № 10 от «07» июня 2017 г.

Заведующий кафедрой программного обеспечения _____ Е.Е.Сурина

1 Методические указания по проведению лекционных занятий

Лекционные занятия в высшем учебном заведении являются основной формой организации учебного процесса и должны быть нацелены на выполнение ряда задач:

- ознакомить студентов со структурой дисциплины;
- изложить основной материал программы курса дисциплины;
- ознакомить с новейшими подходами и проблематикой в данной области;
- сформировать у студентов потребность к самостоятельной работе с учебной, нормативной и научной литературой.

Лекционное занятие представляет собой систематическое, последовательное, монологическое изложение преподавателем-лектором учебного материала, как правило, теоретического характера.

Цель лекции – организация целенаправленной познавательной деятельности студентов по овладению программным материалом учебной дисциплины.

Чтение курса лекций позволяет дать связанное, последовательное изложение материала в соответствии с новейшими данными науки, сообщить слушателям основное содержание предмета в целостном, систематизированном виде.

В ряде случаев лекция выполняет функцию основного источника информации, когда новые научные данные по той или иной теме не нашли отражения в учебниках.

Организационно-методической базой проведения лекционных занятий является рабочий учебный план направления подготовки. При подготовке лекционного материала преподаватель обязан руководствоваться учебными программами по дисциплинам кафедры, тематика и содержание лекционных занятий которых представлена в рабочих программах, учебно-методических комплексах.

При чтении лекций преподаватель имеет право самостоятельно выбирать формы и методы изложения материала, использовать различные технические средства обучения.

Рекомендации по работе студентов с конспектом лекций.

Изучение дисциплины студенту следует начинать с проработки рабочей программы, особое внимание, уделяя целям и задачам, структуре и содержанию курса.

При конспектировании лекций студентам необходимо излагать услышанный материал кратко, своими словами, обращая внимание, на логику изложения материла, аргументацию и приводимые примеры. Необходимо выделять важные места в своих записях. Если непонятны какие-либо моменты, необходимо записывать свои вопросы, постараться найти ответ на них самостоятельно. Если самостоятельно не удалось разобраться в материале, впоследствии необходимо либо на следующей лекции, либо на лабораторном занятии или консультации обратиться к ведущему преподавателю за разъяснениями.

Успешное освоение курса предполагает активное, творческое участие студента путем планомерной, повседневной работы. Лекционный материал следует просматривать в тот же день. Рекомендуемую дополнительную литературу следует прорабатывать после изучения данной темы по учебнику и материалам лекции.

Каждая тема имеет свои специфические термины и определения. Усвоение материала необходимо начинать с усвоения этих понятий. Если какое-либо понятие вызывает затруднения, необходимо посмотреть его суть и содержание в словаре (Интернете), выписать его значение в тетрадь для подготовки к занятиям.

При подготовке материала необходимо обращать внимание на точность определений, последовательность изучения материала, аргументацию, собственные примеры, анализ конкретных ситуаций. Каждую неделю рекомендуется отводить время для повторения пройденного материала, проверяя свои знания, умения и навыки по контрольным вопросам и тестам.

2 Методические указания по лабораторным и практическим работам

Изучение дисциплины «Б.1.В.ДВ.7.2 Разработка и реализация сетевых протоколов» предполагает посещение обучающимися не только лекций, но и лабораторных работ. Лабораторные работы со студентами предназначены для проверки усвоения ими теоретического материала дисциплины.

Основные цели лабораторных работ:

- закрепить основные положения дисциплины;
- проверить уровень усвоения и понимания студентами вопросов, рассмотренных на лекциях и самостоятельно изученных по учебной литературе;
- научить пользоваться нормативной и справочной литературой для получения необходимой информации о конкретных технологиях;
- оказать помощь в приобретении навыков расчета точностных характеристик;
- восполнить пробелы в пройденной теоретической части курса и оказать помощь в его усвоении.

Для контроля знаний, полученных в процессе освоения дисциплины на лабораторных занятиях обучающиеся выполняют задания реконструктивного уровня и комплексное практическое задание.

Целью выполнения задания реконструктивного уровня и комплексного задания студентами является систематизация, закрепление и расширение теоретических знаний, полученных в ходе изучения дисциплины.

Ниже приводятся общие методические указания, которые относятся к занятиям по всем темам:

- в начале каждого лабораторного занятия необходимо сформулировать цель, поставить задачи;
- далее необходимо проверить знания студентами лекционного материала по теме занятий;
- в процессе занятия необходимо добиваться индивидуальной самостоятельной работы студентов;
- знания студентов периодически контролируются путем проведения текущей аттестации (рубежного контроля), сведения о результатах которой доводятся до студентов и подаются в деканат;
- время, выделенное на отдельные этапы занятий, указанное в рабочей программе, является ориентировочным; преподаватель может перераспределить его, но должна быть обеспечена проработка в полном объеме приведенного в рабочей программе материала;
- на первом занятии преподаватель должен ознакомить студентов с правилами поведения в лаборатории и провести инструктаж по охране труда и по пожарной безопасности на рабочем месте;
- преподаватель должен ознакомить студентов со всем объемом лабораторных работ и требованиями, изложенными выше;
- преподаватель уделяет внимание оценке активности работы студентов на занятиях, определению уровня их знаний на каждом занятии.

На лабораторных работах решаются задачи из всех разделов изучаемой дисциплины.

1. Язык программирования Perl.

1.1. Лабораторная работа №1.1. Базовые операторы и типы данных.

В Perl реализованы все обычные арифметические операторы языка C, например (# - символ комментария):

```
$a = 1 + 2;     # Суммируются числа 1 и 2 и сохраняются в $a
$a = 3 – 4;     # Из 3 вычитается 4 и сохраняется в $a
$a = 5 * 6;     # умножаются 5 и 6
$a = 7 / 8;     # 7 делится на 8, в $a будет 0.875
$a = 9 ** 10;   # 9 в 10-й степени
$a = 5 % 2;     # в $a будет остаток от деления 5 на 2
++$a;           # увеличивает на 1 значение $a, затем возвращает новое значение
$a++;           # возвращает значение $a, затем увеличивает его на 1
--$a;           # уменьшает на 1 значение $a, затем возвращает новое значение
$a--;           # возвращает значение $a, затем уменьшает его на 1
```

Дополнительно Perl выполняет следующие операторы:

```
$a = $b . $c;           # объединяет значения переменных $b и $c
$a = $b x $c;   # значение $b повторяется $c раз
```

Примеры операторов присвоения в Perl:

```
$a = $b;        # переменная $b присваивается переменной $a
$a += $b;       # переменная $b суммируется с переменной $a, результат в $a
$a -= $b;       # переменная $b вычитается из переменной $a, результат в $a
$a .= $b;       # переменная $b объединяется с переменной $a, результат в $a
```

Таблица 1. Операторы сравнения строк и чисел.

Операции сравнения	Сравнение строк	Сравнение чисел
Меньше чем	lt	<
Больше чем	gt	>
Равно	eq	==
Меньше чем или равно	le	<=
Больше чем или равно	ge	>=
Не равно	ne	!=

В языке Perl используется три типа данных: скаляры, массивы скаляров и ассоциативные массивы (хэш-массивы).

Скалярный тип данных.

Perl поддерживает три вида значений скалярных переменных: целые числа, числа с плавающей запятой и строки символов.

Имя скалярной переменной состоит из символа $, за которым следует одна буква, затем, выборочно, несколько букв, цифр, символов подчеркивания, например: $x, $my_var, $var2.

Любая скалярная переменная имеет в качестве начального значения пустую строку, "".

Это означает, что не нужно заранее объявлять и определять переменную. Например, выполнение фрагмента кода:

```
$result = $undefined + 2;               # переменная $undefined не определена
print ("Переменная \$result имеет значение $result.\n");
```

выдаст следующий результат:

Переменная $result имеет значение 2.

Примеры:

```
$nine = 9;     # присваивается число 9
$str1 = 'да-да, так оно и было!\n'; # присваивается строка - да-да, так оно и было!
$str2 = "да-да, так оно и было!\n";  # присваивается строка - да-да, так оно и было!
```

Существует два отличия между одинарными и двойными кавычками. Первое отличие в том, что в двойных кавычках переменная заменяется на свое значение, а в одинарных - нет. Второе отличие, в одинарных кавычках не работают управляющие последовательности. Если распечатать предыдущие переменные str1 и str2 то получим:

```
print('Это строка - $str1\n');  # распечатает - Это строка - $str1\n
print("Это строка $str2\n");  # распечатает – Это строка да-да, так оно и было!
```

При необходимости Perl сам, автоматически преобразует строку в целое число, например:

```
$str = "67";
$number = 33;
$result = $str + $number;  # Переменной $result присваивается значение 100.
```

Пример программы преобразующей мили в километры и обратно:

```
#!/usr/bin/perl
print ("Программа преобразования мили в километры и обратно.");
print ("Введите число для преобразования:\n");
$originaldist = <STDIN>;
chop ($originaldist);
$miles = $originaldist * 0.6214;
$kilometers = $originaldist * 1.609;
print ($originaldist, " километров = ". $miles." мили\n");
print ($originaldist, " миль = ". $kilometers." километров\n");
```

Вначале программа выдает сообщение и приглашает ввести число для преобразования.
Выражение $originaldist = <STDIN> присваивает переменной $originaldist строку, читаемую со стандартного устройства ввода (клавиатуры).
Библиотечная функция chop удаляет один символ с правого конца строки символов. При вводе числа с клавиатуры строка чисел завершается символом перевода строки \n.
Выражение chop ($originaldist) удаляет его.
Следующие четыре строки программы присваивают переменным $miles и $kilometers расчетные значения и выдают их на экран.

Массивы скаляров.

Perl позволяет упорядочивать набор скалярных значений в списки.
Примеры списков:
(1, 5.3, "киска", 2) – список, состоящий из чисел 1, 5.3, строки 'киска' и числа 2.
() – такой список называется пустым.
(17, $var1, "учеба") – второй элемент списка – значение скалярной переменной. Если $var1=25, то список будет (17, 25, "учеба").
Списки можно хранить в специальной переменной, называемой массивом.
Массив может быть пустым, хранить пустой список. Максимальный размер массива ограничен доступной памятью. Переменная массив обозначается @<имя_массива>, например:

```
@names = ("Саша", "Петя", "Вася"); #массиву @names присваивается список из трех элементов "Саша", "Петя", "Вася".
```

Доступ к элементам массива производится по их индексу, начинающихся с 0.

```
$name = $names[1]; # присваивает переменной $name значение - Петя.
$name = $names[3];  #при обращении к несуществующему элементу списка возвращается пустая строка, т.е. будет $name=''.
```

В качестве номера элемента можно подставлять переменную, н.п.:
$index = 2;
$name = $names[$index]; # присваивает переменной $name значение – Вася.
Обратите внимание, что обращение к элементу массива происходит как к скалярной переменной $names[индекс]. Узнать индекс последнего элемента массива можно по выражению @#<имя_массива>. У нас @#names=2.
Выражение $size = @names присвоит переменной $size размер массива @names. В нашем примере будет $size = 3, т.к. в массиве три элемента.
Распечатать содержимое массива можно по команде print, например
print("@names\n"); #выведет на экран – Саша Петя Вася

Пример программы, генерирующей случайные целые числа от 1 до 10:

```
1: #!/usr/bin/perl
2:
3: # Сбор набора случайных чисел
4: $count = 1;
5: while ($count <= 100) {
6: $randnum = int( rand(10) ) + 1;
7: $randtotal[$randnum] += 1;
8: $count++;
9: }
10:
11: # Печать количества попаданий каждого случайного числа
12: $count = 1;
13: print ("Подсчет попадании каждого числа:\n");
14: while ($count <= 10) {
15: print ("\tЧисло $count: $randtotal[$count] раз\n");
16: $count++;
17: }
```

Для удобства разбора строки пронумерованы. В рабочем коде номеров нет.
Программа состоит из двух частей. Первая часть (строки 4-9) собирает наборы случайных чисел от 1 до 10. Вторая часть (строки 12-17) печатает количество попадании каждого случайного числа. Строка 5 задает количество циклов равным 100. Строка 6 высчитывает случайное число. Фрагмент кода int(rand(10)) + 1 состоит из двух библиотечных функции. Первая выполняется rand(10) возвращающая случайное число с плавающей запятой от 0 до 1 умноженное на аргумент 10, т.е. числа больше 0 и меньше 10. Вторая функция int удаляет дробную часть и оставляет целую часть полученного числа. После прибавления 1 получаем случайные числа от 1 до 10. Строка 7 заполняет массив с индексами от 1 до 10 количеством попадании случайных чисел. Строки 14 и 15 в цикле распечатывают количество попадании для каждого элемента массива (случайного числа).

Perl позволяет одновременный доступ к нескольким элементам списка. Например, выражение @subarray = @array[5,6,7] создает новый массив @subarray и присваивает ему значения пятого, шестого и седьмого элементов массива @array.
Вместо конкретных индексов можно указывать диапазон индексов. Выражение @subarray = @array[100 .. 200] присваивает @subarray значения с 100-го 200-й элементов @array.

Задание. Напишите программу, которая инициализирует массив @array пятью любыми числами. Присваивает массиву @subarray значение второго и третьего элемента массива @array. Распечатывает на экране содержимое массива @array, затем, на следующей строке содержимое @subarray.

Массиву можно присвоить подстроки из строки, отделенные указанным разделителем с помощью команды split. Например, строки кода:
$string = "слова::разделенные::двоеточиями";
@array = split(/::/,$string);
присвоят массиву @array = ("слова", "разделенные", "двоеточиями")

Пример программы, подсчитывающей слова:

```
1: #!/usr/bin/perl
2:
3: $wordcount = 0;
4: $line = <STDIN>;
5: while ($line ne "") {
6: chop ($line);
7: @array = split(/ /, $line);
8: $wordcount += @array;
9: $line = <STDIN>;
10: }
11: print ("Количество слов в вашем тексте: $wordcount\n");
```

При нажатии на клавиши Ctrl-D (End-of-File), выражение <STDIN> возвращает пустую строку. Строка 5 программы проверяет нажатие Ctrl-D. Простое нажатие на Enter возвращает строку с символом новая строка '\n', т.е. она не пустая. Символ '\n' удаляется командой chop. Сторка 7 программы делит каждую введенную строку на слова и присваивает их массиву @array. Разделитель – пробел, split(/ /,$array). Количество слов в введенной строке будет равен размеру массива @array. Строка 8 программы суммирует общее количество слов во всех введенных с клавиатуры строках. Клавиатура читается до тех пор пока не будут нажаты клавиши Ctrl-D.

Задание. Напишите программы, которые:

1. ***читает с клавиатуры строку чисел разделенных пробелом и:***
   - ***подсчитывает и распечатывает сумму для строки***
   - ***подсчитывает и распечатывает общую сумму со всех строк***
2. ***читает с клавиатуры строки со словами и подсчитывает слова the.***

Массив скаляров широко используется, но имеет один недостаток. Трудно запомнить в каком элементе массива хранятся нужные данные. Иногда, для их нахождения необходимо просмотреть весь массив. Для устранения этого недостатка Perl вводит следующий тип данных.

Ассоциативный массив (хэш-массив).

В отличие от обычного массива, где доступ к элементам массива прозводится по их числовому индексу, доступ к элементам хэш-массива производится по индексу имеющему скалярное значение. Обозначается хэш-массив символом %имя_массива. Например, необходимо поместить в массив возраст всех сотрудников подразделения:

```
%ages = ("Иванов", 39,
         "Петров", 27,
         "Сидоров", 108,
         "Попов", 23);
```

Теперь можно легко определить возраст сотрудника :

```
$ages{"Иванов"}          # возвращает 39
$ages{"Петров"}          # возвращает 27 и т.д.
```

Предыдущее выражение можно переписать в более удобном виде:

```
%ages = ("Иванов" => 39,
         "Петров" => 27,
         "Сидоров" => 108,
         "Попов" => 23);
```

Здесь имена сотрудников являются индексами (ключами) хэш-массива. Их можно извлечь из хэш-массива и присвоить обычному массиву с помощью функции keys, например код:

@names = keys (%ages);

поместит в @names = ("Иванов", "Петров", "Сидоров", "Попов");

Важно отметить, что в хэш-массиве элементы хранятся в случайном порядке, т.е. не так как их проинициализировали. Чтобы получить отсортированный список необходимо использовать функцию sort, например:

@names = sort keys (%ages);

Распечатать полностью хэш-массив можно следующим образом:

```
foreach $name (sort keys (%ages)) {
    print ("$name: $ages{$name}\n");
}
```

Здесь выражение (sort keys (%ages)) возвращает отсортированный список ключей (имена сотрудников), которые поочередно подставляются в качестве значения переменной $name и для каждого такого случая (оператор цикла foreach) на экране распечатывается имя и возраст.

В хэш-массив можно легко добавить, несуществующий ранее, элемент массива, например:

$ages{"Попова"} = 13;

Удалять элемент из массива необходимо функцией delete, например:

delete($ages{"Попова"});

Задание. Напишите программу, которая:

- ***предлагает ввести имена сотрудников с клавиатуры и помещает их в обычный массив;***
- ***для каждого сотрудника из массива предлагает ввести марку их автомобилей и помещает в ассоциативный массив;***
- ***распечатывает полный список сотрудников и их авто.***

1.2. Лабораторная работа №1.2. Управляющие структуры.

В лабораторной работе №1, для организации программных циклов использовалось выражение while, например:

```
$count = 1;
while ($count <= 5) {
    # какие то выражения языка Perl
    $count++;
}
```

Здесь, для организации цикла, требуется выполнить три действия:

- проинициализировать счетчик итерации цикла $count = 1 до начала цикла;
- выполнять проверку условия $count <= 5 при каждом вхождении в цикл;
- увеличивать счетчик итерации $count на 1 в теле цикла (часто забывается).

Эти действия можно выполнить за один раз с использованием оператора for. Тогда предыдущий пример перепишется в следующем виде:

```
for($count = 1; $count <= 5; $count++) {
    # какие то выражения языка Perl
}
```

Иногда требуется, чтобы в цикле, перед каждой итерацией, выполнялись несколько действий, например:

```
#!/usr/bin/perl
        for($line=<STDIN>, $count=1; $count <= 3; $line = <STDIN>, $count++) {
                print("$line");
        }
```

Приведенная программа читает с клавиатуры 4 строки и распечатывает на экране первые три.

В лабораторной работе №1 было задание найти слово the в массиве скаляров, который заполнен с клавиатуры. При помощи for можно написать:

```
        for ($count = 1; $count <= @words; $count++) {
                if ($words[$count - 1] eq "the") {
                print("Нашел таки слово 'the'.\n");
                }
        }
```

Этот код можно упростить с помощью выражения foreach, синтаксис которого следующий:

```
        foreach локальная_переменная (имя_обрабатываемого_массива) {
                #блок_выражений;
        }
```

Тогда, предыдущий пример перепишется в виде:

```
        foreach $word (@words) {
                if ($word eq "the") {
                print("Нашел таки слово 'the'.\n");
                }
        }
```

Здесь, на каждом шаге, в локальную переменную $word помещается очередной элемент массива @words и выполняется проверка условия.

Для обязательного выполнения хотя бы одной итерации цикла используется оператор do с синтаксисом:

```
        do {
                #блок_выражений;
        } while или until (условие);
```

Например, программу чтения клавиатуры пока не получена пустая строка (Ctrl-D) можно переписать следующим образом:

с использованием while

```
do {
        $line = <STDIN>;
} while ($line ne "");
```

с использованием until

```
do {
        $line = <STDIN>;
} until ($line eq "");
```

Задание. Напишите программу, которая:

- ***читает с клавиатуры строки со словами до получения пустой строки (клавиши Ctrl-D);***
- ***подсчитывает количество повторяемых слов и помещает их в хэш-массив;***
- ***выдает на экран информацию о введенных словах и их количестве.***

Поиск по шаблону.

Шаблон – это набор символов, которые необходимо найти в строке. Для углубленного поиска используется язык регулярных выражений, широко поддерживаемый в Perl.

Шаблон помещается между прямыми слэшами

/искомые символы/, н.п. /the/ - поиск слова the.

Для проверки выполнения условия (на истину) используется оператор тестирования =~, например:
$result = $var =~ /abc/; # $result>0 (истина), если в строке $var есть abc, иначе $result=0 (ложь).
Для проверки невыполнения условия (на ложь) используется оператор тестирования !~, например:
$result = $var !~ /abc/; # $result=0, если в строке $var есть abc, иначе $result>0.

Программа, иллюстрирующая поиск по шаблону:

```
#/usr/bin/perl
print ("Задайте вопрос, но только вежливо:\n");
$vopros = <STDIN>;
if ($vopros =~ /пожалуйста/) {
        print ("Спасибо за вежливо заданный вопрос.\n");
} else {
        print ("Вы задали вопрос невежливо.\n");
}
```

В шаблоне можно помещать специальные символы – регулярные выражения.
1) + - одно или более повторений последнего символа, н.п.: /de+f/ # соответствует def, deef, deeeef и т.д.
/[\t]+/ # одно и более повторении знаков табуляции или пробелов.
2) [] – любые символы в квадратных скобках [], н.п.: /ab[cC]d/ # соответствует abcd и abCd
/ab[0123456789]d/ # любая одна цифра между ab и d
/ab[0123456789]+d/ # любой набор цифр между ab и d
В квадратных скобках можно указывать диапазон, через тире, н.п.:
/ab[0-9]d/ # любая одна цифра между ab и d
/ab[0-9]+d/ # любой набор цифр между ab и d
/[A-Z]/# все прописные буквы
/_[a-z]+/ # подчеркивание и любое количество строчных букв
Чтобы исключить появление набора в строке поиска используется символ ^.
/ab[^0-9]d/ # соответствует любому нецифровому символу между ab и d
3) * - ноль и более повторений последнего символа, н.п.:
/de*f/ # соответствует df, def, deef, deeeeeef и т.д.
4) ? - ноль или одно повторение последнего символа, н.п.:
/de?f/ # соответствует df, def, но не deef, deeeeeef и т.д.
5) ^ - искать шаблон в начале строки, н.п.:
/^def/ # строка начинается символами def
6) $ - искать шаблон в конце строки, н.п.:
/def$/ # строка заканчивается символами def
/^def$/ # строка состоит только из символов def
7) | - выбор между двумя шаблонами, н.п.:
/def|abs/ #соответствует набору def или abs

Если необходимо найти набор символов содержащих специальный символ, его нужно записывать со знаком \, н.п.
/*+/ # строка поиска множества символов **********
Пример программы, проверяющей правильность написания переменной в языке Perl:

```
#!/usr/bin/perl
print ("Введите имя переменной: ");
$varname = <STDIN>;
chop($varname);
if($varname =~ /^\$[A-Za-z][_0-9a-zA-Z]*$/) {
```

```
        print("$varname – правильно набранная скалярная переменная\n");
} elsif($varname =~ /^\@[A-Za-z][_0-9a-zA-Z]*$/) {
        print("$varname – правильно набранная переменная типа массив\n");
} elsif($varname =~ /^[A-Za-z][_0-9a-zA-Z]*$/) {
        print("$varname – правильно набранная переменная типа имя файла\n");
} else {
        print("Я не знаю, что это такое $varname\n");
}
```

В приведенном выше примере, необходимо самостоятельно разобрать приведенные регулярные выражения.

Задание 2. Напишите программу, которая:

- ***предлагает ввести слово для поиска;***
- ***читает с клавиатуры строки со словами до получения пустой строки (клавиши Ctrl-D);***
- ***подсчитывает количество повторений запрошенного слова;***
- ***выдает на экран информацию о введенных словах и их количестве.***

Для короткого написания некоторых часто встречающихся диапазонов символов используются ESC-последовательности, которые приведены в таблице 2.

ESC -последовательность	Описание	Диапазон
\d	Любая цифра	[0-9]
\D	Любая нецифра	[^0-9]
\w	Любая буква или цифра	[_0-9a-zA-Z]
\W	Любая небуква и нецифра	[^_0-9a-zA-Z]
\s	Любой пробел	[\r\t\n\f]
\S	Любой непробел	[^\r\t\n\f]

Таблица 2. ESC-последовательности для диапазона символов.

Пример программы, проверяющей правильность ввода целого числа, независимо от знака и системы счисления.

```
#!/usr/bin/perl
print("Введите число: ");
$number = <STDIN>;
chop($number);
if($number =~ /^-?\d+$|^-?0[xX][\da-fA-F]+$/) {
        print("$number – правильное целое число.\n");
} else {
        print("$number – неправильное целое число.\n");

}
```

В вышеприведенном примере, необходимо самостоятельно разобрать регулярные выражения.

Perl предлагает два оператора, которые позволяют произвести замену найденного набора символов на другие символы. Синтаксис первого следующий: s/шаблон/новые_символы/ , н.п.:

```
$string = 'abs123def123';
$string =~ s/123/456/; # меняет 123 на 456, строка $string будет abs456def123.
```

Здесь могут быть использованы опции:

i – игнорировать регистр символов (прописные, строчные).

g – произвести замену для всех совпадений (по умолчанию – только первое совпадение), н.п.:

```
$string =~ s/123/456/g; # меняет 123 на 456, строка $string будет abs456def456.
```

Синтаксис второго следующий: tr/строка1/строка2/

Здесь первый символ строки 1 заменяется на первый символ строки 2, второй символ строки 2 на второй символ строки 2 и т.д., н.п.:

```
$string = "abcdefgh";
$string =~ tr/efgh/abc/; # здесь нет символа для замены h, тогда он будет меняться на
```

последний символ из списка abc. В итоге $string= abcdabcc. Данный оператор, чаще всего, используется для замены строчных букв на прописные или наоборот. Н.п. $line =~ tr/A-Z/a-z/; Здесь в строке $line все прописные буквы меняются на все строчные. При работе с русскими буквами такой способ может не работать. Тогда надо два раза перечислить все 33 буквы русского алфавита, соответственно прописными и строчными.

Задание 3. Напишите программу, которая:

- ***предлагает ввести слово для поиска;***
- ***предлагает ввести слово для замены;***
- ***читает с клавиатуры строки со словами до получения пустой строки;***
- ***находит искомое слово во введенных строках и меняет его на заменяемое слово;***
- ***выдает на экран информацию об исходных и новых строках.***

Задание 4. Напишите программу, которая:

- ***читает с клавиатуры строки со словами до получения пустой строки;***
- ***заменяет во всех словах строчные буквы на прописные;***
- ***выдает на экран информацию об исходных и новых строках.***

Задание 5. Напишите программу, которая:

- ***читает с клавиатуры строки со словами до получения пустой строки;***
- ***находит числа во введенных строках и подсчитывает их количество;***
- ***выдает на экран информацию о количестве найденных чисел.***

Поиск по шаблону по регулярным выражениям языка Perl оказались настолько удачными, что они реализованы и в языке PHP в функциях preg_match() и preg_replace().

Преобразование элементов массива в строку, преобразование строки в массив

Для преобразования элементов массива в строку используется команда join, синтаксис которой следующий:

```
$string = join("символ_разделитель", @array);
```

Здесь все элементы массива @array объединяются в строку $string через символ разделитель. Приведем простой пример:

```
#!/usr/local/bin/perl
@input = <STDIN>;
chop (@input);
$string = join(" ", @input);
```

```
print ("$string\n");
```
Результат работы программы следующий:
```
Это
моя
строка
^D
Это моя строка
```
Для преобразования строки в массив используется команда split, синтаксис которой следующий:
```
@array = split(/символ_разделитель/, $string);
```
Здесь все подстроки строки $string разделенные символом разделителем помещаются в массив @array.
Пример программы, подсчитывающей слова из строк введенных с клавиатуры с использованием команды split приведен в лабораторной работе №1.

Работа с файловой системой.

Базовые команды работы с файлами:
open – открытие файла
close – закрытие файла
print – запись строки в файл
printf – запись форматированной строки в файл
Примеры:
```
open(MYFILE, "/home/st1/1.txt") || die ("Can't open file /home/st1/1.txt");
```
открывает файл /home/st1/1.txt на чтение, в MYFILE- дескриптор открытого файла. При неудаче выдает сообщение об ошибке.
open(MYFILE, ">/home/st1/1.txt") - открывает файл /home/st1/1.txt на запись. Если существовал старый файл /home/st1/1.txt, то он удаляется.
open(MYFILE, ">>/home/st1/1.txt") - открывает файл /home/st1/1.txt на дозапись.
Команду open можно использовать для вызова системных команд. Например команда open(MAIL, "| mail st1") открывает конвейер для отправки почты пользователю st1.
Ниже приведен пример программы открывающей файл пользователей /etc/passwd и распечатывающей список построчно на экран.
```
#!/usr/bin/perl
open (MYFILE, "/etc/passwd") || die ("Can not open file /etc/passwd");
@pass = <MYFILE>;
foreach $line (@pass) {
print("$line");
}
close(MYFILE);  # закрытие файла
```
В приведенной программе используется файл /etc/passwd, где каждая строка содержит данные конкретного пользователя.
Пример использования команды print:
```
print MYFILE1 ("$line");  # записывает строку $line в файл, дескриптор которого
```
MYFILE1, предварительно должен быть открыт на запись.
В вышеприведенном примере файл /etc/passwd содержит списки пользователей сервера server1. Одна строка на одного пользователя по следующей структуре:
имя:пароль:идентификатор пользователя:идентификатор группы:комментарий:домашний каталог:командный интерпретатор
Все данные разделены через разделительный символ :.
Командой slip можно все данные строки присвоить массиву или переменным списка, н.п.:
```
($login,$pass,$id,$gid,$comment,$home,$shell)=split(/:/,$line);
```

Задание 2.1. Напишите программу, которая предлагает ввести имя искомого пользователя. Открывает файл /etc/passwd, находит указанного пользователя и выдает данные о пользователе на экран.
Задание 2.2. Напишите программу, которая ищет всех пользователей, у которых командный интерпретатор не bash. Открывает файл /etc/passwd, находит искомых пользователей, выдает их на экран и записывает их в файл noshell.txt.

1.3. Лабораторная работа №1.3. Perl и база данных MySQL. Прямое подключение к СУБД MySQL и основные команды MySQL.

Предполагается, что студент знаком с основными понятиями и организацией баз данных. Для прямого подключения к СУБД используется клиентская программа mysql. Все примеры приводятся для студента с именем st1 и паролем st1. Другие студенты должны использовать свои имена. Итак, для подключения к СУБД MySQL под именем st1 и паролем st1, необходимо ввести с командной строки ОС команду:
mysql –u st1 –pst1
Обратите внимание, что между ключом –p и паролем st1 нет пробела.
В ответ вы получите приглашение командной строки самой СУБД:
mysql>
где можно вводить DDL и DML команды СУБД. Например, чтобы увидеть список баз данных доступных пользователю st1, необходимо подать команду:
mysql> show databases;
Каждая команда завершается символом ;.
Ответ может быть следующим:

```
+--------------------+
| Database           |
+--------------------+
| information_schema |
| st1                |
+--------------------+
2 rows in set (0.00 sec)
```

Для перехода в выбранную из списка базу данных, например st1, необходимо ввести команду use:
mysql> use st1;
Далее можно просмотреть существующие таблицы в базе данных st1 по команде:
mysql> show tables;
Ответ может быть следующим:

```
+----------------+
| Tables_in_st1  |
+----------------+
| cats           |
| pages          |
| statistic      |
| ticket         |
| users          |
| var_answer     |
+----------------+
6 rows in set (0.00 sec)
```

Новую таблицу, например, с названием users1 можно создать следующим образом, где после набора каждой строки надо нажимать клавишу <Enter>:
mysql> create table users1

```
-> (
-> id int auto_increment not null,
-> login varchar(16) not null,
-> password varchar(16) not null,
-> comment varchar(30),
-> primary key (id),
-> unique (login)
-> );
```

Query OK, 0 rows affected (0.00 sec)

Здесь мы указали поля id – идентификатор пользователя, login – имя пользователя, password – пароль, comment – комментарий. Первичным ключом является поле id с автоувеличением, а уникальным поле login. Поля, помеченные not null являются обязательными к заполнению. По этой команде MySQL создает B-дерево для быстрого поиска нужной строки по полю id.

Чтобы узнать структуру ранее созданной таблицы необходимо подать команду describe:

```
mysql> describe users1;
+----------+-------------+------+-----+---------+----------------+
| Field    | Type        | Null | Key | Default | Extra          |
+----------+-------------+------+-----+---------+----------------+
| id       | int(11)     | NO   | PRI | NULL    | auto_increment |
| login    | varchar(16) | NO   | UNI | NULL    |                |
| password | varchar(16) | NO   |     | NULL    |                |
| comment  | varchar(30) | YES  |     | NULL    |                |
+----------+-------------+------+-----+---------+----------------+
```

4 rows in set (0.00 sec)

В приведенном примере использовались типы данных int и varchar. Для хранения даты и времени используется тип datetime, который структурирован следующим образом “YYYY-MM-DD hh:mm:ss”. Для хранения большого текста используется тип text. Он может хранить до 64Кб текста.

Чтобы ввести нового пользователя необходимо подать команду insert, например:

```
mysql> insert into users1 (login, password, comment)
    -> values ('vova','secret','haha');
```

Query OK, 1 row affected (0.01 sec)

Чтобы просмотреть введенные данные необходимо подать команду select, например:

```
mysql> select * from users1;
+----+-------+----------+---------+
| id | login | password | comment |
+----+-------+----------+---------+
|  1 | vova  | secret   | haha    |
+----+-------+----------+---------+
```

1 row in set (0.00 sec)

Чтобы изменить ранее введенные данные необходимо подать команду update, например:

```
mysql> update users1 set comment='hoho' where comment='haha';
```

Query OK, 1 row affected (0.03 sec)

Rows matched: 1 Changed: 1 Warnings: 0

Новый просмотр выдает:

```
mysql> select * from users1;
+----+-------+----------+---------+
| id | login | password | comment |
+----+-------+----------+---------+
|  1 | vova  | secret   | hoho    |
+----+-------+----------+---------+
```

1 row in set (0.00 sec)
Чтобы удалить запись из таблицы необходимо подать команду delete, например:
mysql> delete from users1 where login='vova';
Query OK, 1 row affected (0.00 sec)
Чтобы удалить таблицу из базы данных необходимо подать команду drop, например:
mysql> drop table users1;
Query OK, 0 rows affected (0.00 sec)
Для выхода из клиентской программы mysql надо подать команду exit:
mysql> exit;
Bye

Задание 3.1. Подключитесь к базе данных и создайте таблицу users1 с полями login,pass,id,gid,comment,home,shell. Поля id и gid типа int,остальные varchar.Из файла /etc/passwd введите нескольких пользователей в базу данных.

Подключение и работа в СУБД MySQL из языка Perl.

Для подключения к базам данных в Perl используется библиотека драйверов DBI, подключить которую можно по команде use DBI.
Далее необходимы сведения о названии базы данных, имя сервера, где установлен MySQL, данные об учетной записи пользователя, н.п. для пользователя st1 с паролем st1:

```
use DBI;
$dsn = 'DBI:mysql:st1:localhost';  # здесь используется драйвер mysql  из библиотеки DBI, имя базы данных st1 и имя пользователя совпадают, так же, поскольку пользовательская программа и MySQL находятся на одной и той же машине можно записать localhost.
$db_user_name = 'st1';
$db_password = 'st1';
$dbh = DBI->connect($dsn, $db_user_name, $db_password) || die "Could not connect to database: $DBI::errstr";
```

По последней команде произойдет подключение к базе данных или появится сообщение об ошибке при неудаче.
Для получения информации из базы данных необходимо подготовить строку запроса.
Например, пользователь ввел имя и пароль для авторизации. Наша программа получила от пользователя имя и пароль и поместила их в переменные $username и $pass. Далее она должна выбрать из базы данных информацию (таблица users1) и проверить правильность, введенного, пароля. Сначала подготавливается строка запроса:

```
$sth = $dbh->prepare(qq{select login, password from users1 where login = '$username'});
```

Затем, подается команда на исполнение:

```
$sth->execute();
```

Результат находится во внутреннем массиве драйвера, называемом курсор. Чтобы получить их в программу необходимо подать команду:

```
($login, $password) = $sth->fetchrow_array();
```

Запрос к базе данных завершается командой:

```
$sth->finish();
```

Далее проверяем правильность пароля:

```
if ($pass eq $password)
{
   ... # вход удачный!!!
}
```

Завершается работа с базой данных командой:

```
$dbh->disconnect ();
```

В рассмотренном выше примере обращение к базе данных вернула одну строку, поскольку поле login является уникальным. Если возвращается несколько строк, необходимо команду $sth->fetchrow_array() использовать в цикле, н.п.

```
$sth = $dbh->prepare(qq{select * from users1});
```

```
$sth->execute();
while (($id, $login, $comment) = $sth->fetchrow_array()) {
    print "$id, $login, $comment\n";
}
$sth->finish();
```

Ввод новой строки в базу данных не возвращает результат, поэтому, вместо команды prepare, а затем execute, можно использовать метод do, н.п.

```
$dbh->do("insert into users1 (login, password, comment) values ('vasya', 'nesecret', 'hihi')");
```

Для отключения от базы данных необходимо подать команду:

```
$dbh->disconnect();
```

Задание 3.2. Напишите программу, которая открывает файл /etc/passwd и заносит все данные о пользователях в таблицу user1 базы данных.

Задание 3.3. Напишите программу, которая предлагает ввести имя искомого пользователя. Обращается к базе данных, таблица users1, находит указанного пользователя и выдает данные о пользователе на экран.

Задание 3.4. Напишите программу, которая ищет всех пользователей, у которых командный интерпретатор не bash. . Обращается к базе данных, таблица users1, находит искомых пользователей, выдает их на экран и записывает их в файл noshell.txt.

2. Web программирование на основе технологии CGI.

Вводная часть.

Web-сайты, построенные с использованием языков html, xhtml и т.д., не позволяют формировать динамические страницы в зависимости от конкретной ситуации. Н.п. выбрать товар из предложенного перечня, отложить товар в корзину и т.д. Для этого используют различные Web -технологии, из которых наибольшее распространение получили CGI, PHP и AJAX. Предполагается, что студент знаком с теоретическими основами CGI, которые рассматриваются в лекционном курсе. Здесь отметим общие практические замечания по написанию CGI-программ.

1. Каждый студент имеет на сервере лабораторного класса свой виртуальный хост, который зарегистрирован в службе DNS класса. Доменное имя в лаборатории имеет название ogti.loc. Если инициалы студента по фамилии и имени английскими буквами bm, и он из группы ПОВТ-08, то его краткое доменное имя будет bm08. С этим именем студент подключает сетевые диски и получает telnet сеанс на сервере. Полное доменное имя будет bm08.ogti.loc. Обращаться к его Web-сайту необходимо http://bm08.ogti.loc, т.е. по протоколу http. Хотя такой подход применяется ко всем студентам, необходимо уточнить у преподавателя своё доменное имя, т.к. могут быть одинаковые инициалы.

2. HTML-файлы помещаются в корневом каталоге сайта, определенном в конфигурационном файле Web. Подход одинаковый для всех студентов, н.п. для студента bm08 это каталог /home/bm08/public_html. Здесь он помещает html-файлы. В этом каталоге есть подкаталоги: pic – для помещения файлов изображения gif, jpg и т.д. размером примерно 100x100 пикселей, cgi-bin – для помещения исполняемых cgi-файлов, lib – для помещения, в дальнейшем, java-скриптов. У конкретного студента вместо bm08 будет своё имя.

3. У каждого студента есть своя база данных в СУБД Mysql на сервере server1, название которого совпадает с логином, н.п. bm08. Подключаться к базе данных необходимо через утилиту mysql, получив предварительно telnet доступ на сервере server1.

4. При написании CGI-программ необходимо помнить следующие два правила:

- сайт размещается на UNIX машине, т.е., поскольку cgi-файлы исполняются на стороне сервера, то они должны быть сохранены в UNIX формате. Как правило, исходный код программ набирают на Windows машине, то при сохранении на сервер надо указать формат сохранения. Это можно сделать во встроенном редакторе программы FAR клавишами Shift-F2. Или в других редакторах, н.п. Notepad++ (не путать с обычным Notepad).

- чтобы cgi-файл был доступен для исполнения web-сервером, необходимо установить соответствующие права доступа на исполнение для псевдопользователя nobody. Поскольку студенты в классе имеют telnet доступ к серверу, то это достигается подачей команды chmod., н.п. есть файл test1.cgi. Надо подать команду

chmod 777 test1.cgi

Если нет telnet доступа, н.п. при размещении хоста на стороннем сервере, для смены прав доступа используют FTP-соединение. В файловом менеджере FAR есть удобные средства для FTP-подключения и смены атрибутов файла.

Рекомендуется, до тестирования сайта, сначала попробовать запустить cgi-программу с командной строки. Если ошибок нет, то тестировать сайт.

Внимание! Если студент работает на домашнем компьютере с использованием инструментов разработки Denwer, то вышесказанное верно при загрузке работающего в Denwer сайта на сервер лабораторного класса. В самом Denwer, если его установка выполнена стандартно на диск C, то формируется виртуальный диск Z, где пути к файлам можно указывать в UNIX стиле. Тогда, н.п. для виртуального хоста localhost, html-файлы надо помещать в каталоге Z:\home\localhost\www, а cgi-файлы в каталоге Z:\home\localhost\cgi. Менять права доступа при этом не надо. Достаточно в начале

cgi-файла изменить строку #!/usr/bin/perl на путь, где находится интерпретатор perl на локальной машине в UNIX стиле. По умолчанию будет #!/usr/local/miniperl/miniperl. Для выполнения первой лабораторной работы такой подход достаточен, но для дальнейших работ потребуются дополнительные модули Perl, и miniperl будет недостаточен. Тогда надо установить ActivePerl и пути поменять. По всем дополнительным вопросам необходимо обращаться к преподавателю.

2.1. Лабораторная работа №2.1. Ознакомительное CGI-программирование с использованием обычных команд языка Perl. Web-страница для регистрации пользователя на сайте.

Задание 1. Необходимо написать web-страницу, предлагающей заполнить поля ввода, фамилию, имя, адрес, телефон и т.д. После заполнения формы, по нажатию кнопки «Зарегистрироваться» данные передаются на сервер, где записываются в файл.

Исходный код главной страницы. Все пояснения приведены в комментариях исходного кода. Основной файл называется index.html.

```
#Здесь все должно быть знакомо – стандартное начало html-документа
<html>
<head><title>Знакомство с технологией CGI</title></head>
<body>
<hr noshade>
<center>
<caption> <h3>Регистрационная форма </h3></caption>
#Здесь открываем форму. Заполненные в форме данные передадутся на сервер по методу
# GET. Получает и обрабатывает данные из формы, cgi-программа test1.cgi на сервере
<form method=get action="/cgi-bin/test1.cgi">
#Поля ввода информации для регистрации организуем в виде таблицы
<table border=0 width=50%>
<th align=left> Имя
<th align=left> Фамилия <tr>
<td> <input type=text size=20 maxlength=30 name="first">
<td> <input type=text size=20 maxlength=30 name="last"> <tr>
<th align=left colspan=2> Улица, дом, квартира <tr>
<td colspan=2>
<input type=text size=80 maxlength=80 name="street"> <tr>
<th align=left> Город
<th align=left> Страна <tr>
<td> <input type=text size=20 maxlength=30 name="city">
<td> <input type=text size=20 maxlength=30 name="state"> <tr>
<th align=left> Индекс
<th align=left> Номер телефона <tr>
<td> <input type=text size=6 maxlength=10 name="ind">
<td> <input type=text size=15 maxlength=15 name="phone" value="(999) 999-999"> <tr>
#В конце добавляем две кнопки, "submit"-выполнить и "reset"-сброс для нового ввода
#По нажатию "submit" данные из окон ввода кодируются и передаются на сервер
<td > <input type="submit" name="reg" value="Зарегистрироваться">
<td > <input type=reset> <tr>
</table>
</form>
</center>
<hr noshade>
</body>
</html>
```

Исходный код файла test1.cgi. Все пояснения приведены в комментариях исходного кода. Перед отладкой этого скрипта, в каталог cgi-bin необходимо скопировать библиотечный файл cgi-lib.pl, если его там нет.

```
#!/usr/bin/perl
require("cgi-lib.pl"); #подключаем библиотеку cgi-lib.pl
&ReadParse(*input); #из этой библиотеки вызываем функцию ReadParse, которая
#декодирует полученные от клиента данные и помещает в хэш-массив input.
 $targetfile = "reg.html"; #выбираем имя файла, куда будем писать учетные записи в
#формате html, файл reg.html будет находиться в том же каталоге, где и test1.cgi
 open (NAMEFILE, ">>$targetfile"); #открываем файл, с возможностью дозаписи
#записи в формате html
 print NAMEFILE "<h3>Имя: ",$input{'first'},"</h3>\n";
 print NAMEFILE "<h3>Фамилия: ",$input{'last'},"</h3>\n";
 print NAMEFILE "<h3>Город: ",$input{'city'},"</h3>\n";
 print NAMEFILE "<h3>Страна: ",$input{'state'},"</h3>\n";
 print NAMEFILE "<h3>Индекс: ",$input{'ind'},"</h3>\n";
 print NAMEFILE "<h3>Телефон: ",$input{'phone'},"</h3>\n";
 print NAMEFILE "<p><hr><p>\n";
 close (NAMEFILE); #закрываем открытый файл, и передаем клиенту результат
 print &PrintHeader; #сначала передаем http-заголовок, функция PrintHeader из cgi-lib
 print <<"end_msg"; #далее передаем html-текст до метки end_msg
 <html>
 <head><h2>Результат выполнения</h2></head>
 <hr>
 <h3> Регистрационные данные товарища $input{'first'} $input{'last'} записаны в файл
$targetfile</h3>
 <hr>
 </html>
end_msg        #вот метка end_msg, она должна стоят в начале строки без отступа
print "\n";     #завершаем все пустой сторой.
```

Предложенные два файла набрать, сохранить в каталоги указанные в пункте 2 вводной части и отладить набирая http://bm08.ogti.loc , вместо bm08 подставить свое имя.

Задание 2. Необходимо переписать вышеприведенную web-страницу, где будет выполняться та же задача, но учетные данные будут записываться не в файл, а в базу данных Mysql.

Для выполнения задания необходимо в perl установить модуль DBI, где есть драйверы для подключения к базам данных. На лабораторном сервере все установлено. *При работе на домашнем компьютере надо установить ActivePerl, затем с помощью утилиты ppm установить модуль DBI.*

В базе данных создать таблицу reg с полями со следующей командой:

```
create table reg (first varchar(30), last varchar(30), street varchar(80), city varchar(30), state
varchar(30), ind varchar(10), phone varchar(15));
```

В исходном тексте файла index.html меняется только одна строка:

```
<form method=get action="/cgi-bin/test2.cgi">
```

т.е. вместо test1.cgi теперь будет запускаться программа test2.cgi. Вот его исходный код:

```
#!/usr/bin/perl
use DBI;       #Подключаем установленный модуль DBI
require("cgi-lib.pl");
&ReadParse(*input);
#Подключаемся к базе данных, вместо bm08 записать своё имя
$db=DBI->connect("dbi:mysql:bm08","bm08","bm08")||die($DBI::errstr."\n");
```

```
$db->{AutoCommit}=0;
$db->{RaiseError}=1;
#Подготавливаем строку запроса – запись в БД принятых данных
$query="insert into reg (first,last,street,city,state,ind,phone) values
('$input{'first'}','$input{'last'}','$input{'street'}','$input{'city'}','$input{'state'}','$input{'ind'}','$in
put{'phone'}')";
$sth=$db->prepare($query) or die $sth->errstr;
$sth->execute or die $sth-errstr;       #Выполняем запрос на вставку
print &PrintHeader;
 print <<"end_msg";
 <html>
 <head><h2>Результат выполнения</h2></head>
 <hr>
 <h3> Регистрационные данные товарища $input{'first'} $input{'last'} записаны в файл
$targetfile</h3>
 <hr>
 </html>
end_msg
print "\n";
$sth->finish();
$db->disconnect();    #Закрываем соединение с БД
```

Предложенные два файла набрать, сохранить в каталоги указанные в пункте 2 вводной части и отладить набирая http://bm08.ogti.loc , вместо bm08 подставить свое имя.

2.2. Лабораторная работа №2.2. CGI-программирование с использованием модуля CGI языка Perl.

Web-страница предлагающая готовые конфигурации компьютеров.

Задание. Необходимо написать web-страницу, предлагающей готовые конфигураций компьютеров от известных фирм. В зависимости от выбранного варианта сообщается цена компьютера.

Те, кто работает на домашних компьютерах, должны установить модуль CGI.

2.1.1. **Исходный код главной страницы.** Все пояснения приведены в комментариях исходного кода. Основной файл называется lab2.pl. (Расширение .pl выбран для совместимости с теми, кто работает на Denwer). В работе обратите внимание на то, что в тэге <forms> нет атрибута action, т.е. не указываем какая программа обрабатывает данные, заполненные в форме. В таких ситуациях программа, создающая динамическую страницу, всегда вызывает сама себя, т.е. lab2.pl. Исходный код следующий:

```
#!/usr/bin/perl
require "mylib2.pl";   #подключаем библиотеку со своими функциями
use CGI;                        #подключаем модуль CGI
$q = new CGI;                   #создаем новый объект CGI, в $q -указатель на объект

&top_out($q);          #вызываем функцию, создающую верхнюю часть страницы
if($q->param()) {               #если web-форма заполнена, т.е. компьютер выбран
        &mycalc2($q);           #вызываем функцию расчета стоимости и вывода на экран
        &end_out($q);           #вызываем функцию, создающую нижнюю часть страницы
        exit;                   #выход из сценария
}                               #если web-форма не заполнена, т.е иначе
&myform2($q);                   #вызываем функцию вывода на экран web-форму для
заполнения
&end_out($q);                   #вызываем функцию, создающую нижнюю часть страницы
```

Теперь перейдем к своей пользовательской библиотеке mylib2.pl, куда помещаем все наши функции в форме:

```
sub имя_функции {
        исходный код функции
}
.
.
.
sub имя_следующей_функции {
        исходный код следующей функции
}
1;      #Это метка конца библиотечного файла.
```

2.1.2. **Исходный код библиотечного файла mylib2.pl.** Все пояснения приведены в комментариях исходного кода.

```
sub myform2 {                   #первая функция myform2, выдает web-форму для заполнения
    my ($q) = @_;    #от вызывающего скрипта получаем указатель на объект CGI

     print $q->start_form(-name => 'main2').        #создаем web-форму с именем main2
                        $q->center.                         #равняем по центру
#создаем таблицу со следующими атрибутами
                        $q->start_table({-width=>'60%', -border=>5, -bgcolor=>"#bbbbaa"}).
#создаем строку таблицы с заголовком в 3 столбца и помещаем туда текст. Эти 2 строчки должны #быть написаны одной строкой
                        $q->Tr($q->th({-colspan=>'3'}, $q->h3("Выберите свой любимый компьютер")));
#создаем строку таблицы с данными: в 1 столбце картинка, во 2 столбце фирма, в3 - кнопка
                print $q->Tr(
                        $q->td($q->img({-src=>'/pic/comp1.jpg'})),
                        $q->td('Компьютер от фирмы <br>Hewlett Packard'),
                        $q->td($q->submit(-name=>'hp', -value => 'Выбрать')));
#создаем строку таблицы с данными: в 1 столбце картинка, во 2 столбце фирма, в3 - кнопка
                print $q->Tr(
                        $q->td($q->img({-src=>'/pic/comp2.jpg'})),
                        $q->td('Компьютер от фирмы <br>Dell Computers'),
                        $q->td($q->submit(-name=>'dell', -value => 'Выбрать')));
#создаем строку таблицы с данными: в 1 столбце картинка, во 2 столбце фирма, в3 - кнопка
                print $q->Tr(
                        $q->td($q->img({-src=>'/pic/comp3.jpg'})),
                        $q->td('Компьютер от фирмы <br>ASUS Computers'),
                        $q->td($q->submit(-name=>'asus', -value => 'Выбрать')));
    print $q->end_table;    #конец таблицы
    print $q->end_form;     #конец формы
}

#следующая функция mycalc2, выдает сообщение о стоимости выбранного компьютера, в #зависимости от нажатой кнопки
```

```
sub mycalc2 {
    my ($q) = @_;    #от вызывающего скрипта получаем указатель на объект CGI
        my @comp = $q->param();          #получаем параметр модуля CGI, в нашем
примере там #только один параметр, имя нажатой кнопки(п.п.hp) = значение нажатой
кнопки (п.п. Выполнить)
#далее в зависимости от имени нажатой кнопки формируем: цену, имя фирмы-
производителя, #конфигурацию компьютера
        if($comp[0] eq 'hp') {
                $price = 2000;
                $comp_name = 'Hewlett Packard';
                $conf = 'CPU P-IV 2.8GHz/RAM 4Gb/HDD 300Gb';
        } elsif($comp[0] eq 'dell') {
                $price = 1800;
                $comp_name = 'Dell Computers';
                $conf = 'CPU AMD 2.5GHz/RAM 2Gb/HDD 200Gb';
        } elsif($comp[0] eq 'asus') {
                $price = 1000;
                $comp_name = 'ASUS Computers';
                $conf = 'CPU Celeron 2.4GHz/RAM 1Gb/HDD 100Gb';
        }
# далее выводим сообщение на экран по сценарию аналогично вышеприведенной
        print
                $q->start_form.
                $q->center.
#следующие 2 строчки должны быть написаны одной строкой
                $q->h3("Вы выбрали прекрасный компьютер от фирмы $comp_name<br> с
конфигурацией $conf всего за \$$price<br>").
                $q->submit(-value => 'OK').
                $q->end_form.
                $q->end_html;
}
#следующая функция top_out, выдает верхнюю часть (шапку) страницы. Эта функция
всегда #вызывается первой из основного скрипта. Поэтому она посылает клиенту http-
заголовок и #открывает тэг HTML
sub top_out {
    my ($q) = @_;

        print
# посылаем клиенту http-заголовок
                $q->header(-charset=>'windows-1251').
# открываем тэг HTML с соответствующим титулом
                $q->start_html(-title=>'Компьютерный супермаркет!', -bgcolor=>"#cccccc").
# выравниваем по центру
                $q->center.
# создаем таблицу
                $q->start_table({-border=>0, -bgcolor=>"#bbccaa"}).
# создаем строку с картинкой и вывеской. Эти 2 строки кода надо писать одной строкой
до точки
                $q->Tr($q->td($q->img({-src=>'/pic/lap.jpg'})),$q->td($q-
>h1(' ',' ','Магазин-Салон: Компьютеры и комплектущие',' '))).
                $q->end_table.          # конец таблицы
                $q->p.          # следующий параграф
```

```
        $q->hr();               # подводим черту
}

#последняя функция end_out, выдает нижнюю часть страницы. Здесь закрываем тэг HTML
sub end_out {
    my ($q) = @_;

        print $q->hr;
        print $q->div({-align=>'right'},"My Web Form");
        print $q->end_html;
}
1;              #метка конца библиотечного файла
```

Пробуем и отлаживаем работу страницы, набирая http://bm08.ogti.loc/cgi-bin/lab2.pl
Вместо bm08 необходимо поставить свое имя.

Web-страница предлагающая собрать компьютер из комплектующих компонентов.

Задание. Необходимо написать web-страницу, предлагающей пользователю самому собрать компьютер из предлагаемых комплектующих компонентов. В зависимости от выбранных компонентов сообщается цена компьютера. Форма построения работы, аналогична предыдущей.

Исходный код главной страницы. Файл называется lab3.pl. Код аналогичен предыдущей работе. Отличие, подключается файл mylib3.pl, вызываются функции myform3 и mycalc3.

```
#!/usr/bin/perl
require "mylib3.pl";
use CGI;
        $q = new CGI;
        &top_out($q);
        if($q->param()) {
                &mycalc3($q);
                &end_out($q);
                exit;
        }
        &myform3($q);
        &end_out($q);
```

Исходный код библиотечного файла mylib3.pl. Все пояснения приведены в комментариях исходного кода.

```
sub myform3 {       #первая функция myform3, выдает web-форму для заполнения
    my ($q) = @_;
#составляем хэш-массивы для радио кнопок radio_group и выпадающих меню popup_menu
#в них коротким наименованиям соответствуют длинные названия компонентов
#первый хэш – для процессоров. Эти 2 строчки надо писать одной строкой
        my %cpu=('p32'=>'Intel Pentium4 3.2GHz','c28'=>'Intel Celeron
2.8GHz','amd25'=>'AMD [Box]  2.5GHz');
#второй хэш – для памяти
        my %ram=('sam4'=>'Samsung 4Gb','gig1'=>'Gigabyte 1Gb','as05'=>'ASUS 512Mb');
#следующий хэш – для жестких дисков. Так же одной строкой
        my %hdd=('sg500'=>'Seagate 500Gb','sg200'=>'Seagate 200Gb','sam100'=>'Samsung
100Gb');
#следующий хэш – для мониторов. Так же одной строкой
```

```
        my %monitor=('flat19'=>'LCD Flatron L1919S','saml19'=>'Samsung LCD
19inch','samc17'=>'Samsung CRT 17inch');
#следующий хэш – для DVD-приводов. Так же одной строкой
        my %drive=('neccdrw'=>'NEC CDRW','fuddvd'=>'Fudjitsu DVD-ROM','lgdvd'=>'LG
DVD-RW');
#стартуем форму
      print $q->start_form(-name => 'main3').
                        $q->center.
#стартуем таблицу с атрибутами
                        $q->start_table({-width=>'60%', -border=>10, -bgcolor=>"#bbbbaa"}).
#первая строка таблицы заголовочная
                        $q->Tr($q->th({-colspan=>'2'}, $q->h3("Составьте свою конфигурацию
компьютера из <br> предлагаемых компонентов")));
#следующая строка таблицы – пошли данные в 2 столбца. В 1 столбце название
компоненты, #во 2 – выбор с помощью радио кнопок. Атрибуты 1(-name)-имя радио
группы, 2(-values)-#имена кнопок из соответствующего хэш-массива, 3(-default)-кнопка
выбранная по умолчанию, #4(-labels)- показные длинные названия из соответствующего
хэш-массива
                print $q->Tr(
                $q->td('Процессор'),
                $q->td($q->radio_group(-name=>'cpu', -values=>[keys %cpu],-default=>'p32', -
linebreak=>'true', -labels=>\%cpu)));
#не забывайте писать 2 строки подобные предыдущей – в одну строчку
#следующая строка таблицы – аналогична предыдущей, только для памяти
                print $q->Tr(
                $q->td('Память ОЗУ'),
                $q->td($q->radio_group(-name=>'ram', -values=>[keys %ram],-default=>'gig1', -
linebreak=>'true', -labels=>\%ram)));
#следующая строка таблицы – аналогична предыдущей, только для HDD, но с
использованием #popup_menu, атрибуты те же самые.
                print $q->Tr(
                $q->td('Жесткий диск'),
                $q->td($q->popup_menu(-name=>'hdd', -values=>[keys %hdd],-default=>'lg1', -
linebreak=>'true', -labels=>\%hdd)));
#следующая строка таблицы – аналогична предыдущей, только для монитора
                print $q->Tr(
                $q->td('Монитор'),
                $q->td($q->popup_menu(-name=>'monitor', -values=>[keys %monitor],-
default=>'flat19', -linebreak=>'true',-labels=>\%monitor)));
#следующая строка таблицы – аналогична предыдущей, только для DVD-привода
                print $q->Tr(
                $q->td('Привод'),
                $q->td($q->popup_menu(-name=>'drive', -values=>[keys %drive],-
default=>'fuddvd', -linebreak=>'true',-labels=>\%drive)));
#следующая строка таблицы, с помощью чекбоксов, дополнительно предлагаются модем
и колонки
                print $q->Tr(
                $q->td('Дополнительно'),
                $q->td($q->checkbox(-name=>'sound', -checked=>1, -value=>'true', -
label=>'Мультимедиа'),
                $q->checkbox(-name=>'modem', -checked=>1, -value=>'true', -label=>'ADSL
модем'))
```

```
                );
#следующая строка таблицы – завершающая с кнопкой Submit
        print $q->Tr(
          $q->td({-colspan=>'2'},$q->submit(-value => 'Посчитать стоимость')));
        print $q->end_table;      #конец таблицы
        print $q->end_form;       #конец формы
}
#следующая функция mycalc3, выдает сообщение о стоимости собранного компьютера, в
#зависимости от набранных компонентов
sub mycalc3 {
      my ($q) = @_;
#здесь получаем от параметров объекта CGI – короткие имена выбранных компонентов
          my $cpu=$q->param('cpu');
          my $ram=$q->param('ram');
          my $hdd=$q->param('hdd');
          my $monitor=$q->param('monitor');
          my $drive=$q->param('drive');
          my $sound=$q->param('sound');
          my $modem=$q->param('modem');
#далее анализируем то, что получили и формируем длинные названия и общую цену
          if($cpu eq 'amd25') {
                  $cpu = 'AMD Sempron 2500+ Palermo Soc754[Box]';
                  $price = 250;
          } elsif($cpu eq 'c28') {
                  $cpu = 'Celeron D2800 FSB533 256k';
                  $price = 100;
          } elsif($cpu eq 'p32') {
                  $cpu = 'Pentium P-IV 3.2GHz Soc775 1024k FSB 800';
                  $price = 350;
          }
          if($ram eq 'sam4') {
                  $ram = 'Samsung DDR 4Gb';
                  $price += 200;
          } elsif($ram eq 'gig1') {
                  $ram = 'Gigabyte DDR 1Gb';
                  $price += 100;
          } elsif($ram eq 'as05') {
                  $ram = 'ASUS DDR 512 Mb';
                  $price += 50;
          }
          if($hdd eq 'sg500') {
                  $hdd = 'Seagate ST350082 500Gb';
                  $price += 100;
          } elsif($hdd eq 'sg200') {
                  $hdd = 'Seagate ST320082 200Gb';
                  $price += 80;
          } elsif($hdd eq 'sam100') {
                  $hdd = 'Samsung S80233 100Gb';
                  $price += 70;
          }
          if($monitor eq 'flat19') {
                  $price += 250; $monitor = 'LCD Flatron L1919S';
```

```
        } elsif($monitor eq 'saml19') {
                $price += 350; $monitor = 'Samsung LCD 19inch';
        } elsif($monitor eq 'samc17') {
                $price += 100; $monitor = 'Samsung CRT 17inch';
        }
        if($drive eq 'lgdvd') {
                $price += 150; $drive = 'LG DVD-RW';
        } elsif($drive eq 'fuddvd') {
                $price += 100; $drive = 'Fudjitsu DVD-ROM';
        } elsif($drive eq 'neccdrw') {
                $price += 60; $drive = 'NEC CDRW';
        }
        if($sound) {
                $price += 10; $sound = 'Стереоквадроколонки Тарам-Парам';
        } else {
                $sound = 'нет';
        }
        if($modem) {
                $price += 100; $modem = 'ADSL модем ZTE-391';
        } else {
                $modem = 'нет';
        }
#далее все данные передаем на браузер клиента в виде таблицы
        print
                $q->start_form.                 #стартуем форму
                $q->center.
#стартуем таблицу с указанными атрибутами
                $q->start_table({-width=>'60%', -border=>5, -bgcolor=>"#bbbbaa"}).
#формируем первую строку таблицы - заголовок
                $q->Tr($q->th({-align=>'center'}, $q->h3("Вы сделали потрясающий
выбор!<br>Компьютер с превосходной комплектацией!<br>Всего за \$$price!"))).
#формируем следующую строку таблицы – данные по пунктам
                $q->Tr($q->td({-align=>'left'},
                $q->ul($q->li(["Процессор: $cpu","Оперативная память: $ram","Жесткий
диск: $hdd", "Монитор: $monitor", "CD-DVD привод: $drive", "Мультимедиа: $sound",
"Модем: $modem"])))).
#формируем следующую строку таблицы – кнопка для возврата обратно
               $q->Tr($q->td({-align=>'center'},'Вы очень довольны?<br>',$q->submit(-value =>
'Yes!'))).
                $q->end_table.
                $q->end_form;
}
#следующая функция top_out, выдает верхнюю часть (шапку) страницы. Точная копия из
mylib2.pl
sub top_out {
    my ($q) = @_;
        print
                $q->header(-charset=>'windows-1251').
                $q->start_html(-title=>'Компьютерный супермаркет!', -bgcolor=>"#cccccc").
                $q->center.
                $q->start_table({-border=>0, -bgcolor=>"#bbccaa"}).
```

```
                $q->Tr($q->td($q->img({-src=>'/pic/lap.jpg'})),$q->td($q-
>h1(' ',' ','Магазин-Салон: Компьютеры и комплектущие',' '))).
                $q->end_table.
                $q->p.
                $q->hr();
}
#последняя функция end_out, выдает нижнюю часть страницы. Здесь закрываем тэг HTML
sub end_out {
      my ($q) = @_;
         print $q->hr;
         print $q->div({-align=>'right'},"My Web Form");
         print $q->end_html;
}
1;              #метка конца библиотечного файла
```

Пробуем и отлаживаем работу страницы, набирая http://bm08.ogti.loc/cgi-bin/lab3.pl
Вместо bm08 необходимо поставить свое имя.

3. Web программирование на основе технологии PHP.

3.1. Лабораторная работа №3.1. WEB-программирование с использованием языка PHP.

Web-страница предлагающая готовые конфигурации компьютеров.

Задание. Необходимо переписать на языке PHP web-страницу, предлагающей готовые конфигураций компьютеров от известных фирм ранее написанную на языке Perl. В зависимости от выбранного варианта сообщается цена компьютера.

Все php-файлы помещаются там же, где и html-файлы. Никакие права доступа на исполнение ставить не надо, поскольку Web-сервер сам интерпретирует php-скрипты.

Исходный код главной страницы. Все пояснения приведены в комментариях исходного кода. Основной файл называется lab4.php. Исходный код следующий:

```
<?php                       #метка открытия PHP-скрипта
session_start();            #открываем сессию
include "mylib4.php";       #подключаем библиотеку с функцией
require "top_out.php";              #запускаем скрипт top_out.php – отображаем верхнюю
часть
$count = count($_POST);     #подсчитываем количество параметров переданных скрипту
по #методу POST от заполненной WEB-формы, если нажали на кнопки $count>0
if($count > 0) {
#если нажатая кнопка не submit, то вызываем функцию mycalc4 из библиотеки mylib4.php
и передаем #ей полученные параметры формы $_POST
      if(!isset($_POST['submit'])) {
            mycalc4($_POST);    #подсчитываем сумму и выдаем сообщение, где для
#возврата надо нажать на кнопку OK, которое имеет название name=submit
            exit;
      } else {          #иначе, т.е. нажата кнопка submit, то
            require "myform4.php";              #вызываем скрипт отображающая
основную форму
            require "end_out.php";              # вызываем скрипт отображающая
нижнюю часть
            exit;
      }
}else {           #иначе, т.е. в первый раз зашли на страницу
```

```
require "myform4.php";                  #вызываем скрипт отображающая основную форму
require "end_out.php";                  # вызываем скрипт отображающая нижнюю часть
страницы
exit;
}
?>
```

Исходный код библиотечного файла mylib4.php. Все пояснения приведены в комментариях исходного кода.

```
<?                          #метка открытия PHP-скрипта
function mycalc4($b) {              #объявление функции, в $b – указатель на переданные
параметры
        if(isset($b['hp'])) {           #если была нажата кнопка выбора hp
                $price = 2000;#ставим цену
                $comp_name = 'Hewlett Packard';    #имя производителя
                $conf = 'CPU P-IV 2.8GHz/RAM 4Gb/HDD 300Gb';       #конфигурация
        } elseif(isset($b['dell'])) {     #если была нажата кнопка выбора dell
                $price = 1800;
                $comp_name = 'Dell Computers';
                $conf = 'CPU AMD 2.5GHz/RAM 2Gb/HDD 200Gb';
        } elseif(isset($b['asus'])) {     #если была нажата кнопка выбора asus
                $price = 1000;
                $comp_name = 'ASUS Computers';
                $conf = 'CPU Celeron 2.4GHz/RAM 1Gb/HDD 100Gb';
        }
#далее выдаем радостное сообщение
echo '<html>';
echo    '<form method="post" action="lab4.php" enctype="multipart/form-data">';
echo    '<center /><h3>Вы выбрали прекрасный компьютер от фирмы '.$comp_name.' <br> с
конфигурацией '.$conf.' всего за '.$price.' <br></h3>';
echo    '<input type="submit" name="submit" value="OK" />';
echo    '</form>';
echo    '</html>';
#далее запишем выбранный компьютер в корзину, хранящейся в файле сессии
#проверяем есть ли в сессии параметр BASKET, если нет то создаем и присваиваем
пустой массив
if (!isset($_SESSION['BASKET'])) $_SESSION['BASKET']=array();
#формируем строку для записи в BASKET
$item = @array(sname => 'Компьютер', lname => $comp_name, price => $price);
#помещаем данные о выбранном товаре в BASKET
array_push($_SESSION['BASKET'],$item);
return;         #возврат из функции
}
?>
```

Исходный код файла top_out.php. Все пояснения приведены в комментариях исходного кода.

```
<?              #если есть корзина, подсчитываем количество выбранных товаров
if (isset($_SESSION['BASKET'])) {
        $count = count($_SESSION['BASKET']);
        } else { $count = 0; }
?>
```

```
#далее идет отображение верхней части страницы в виде HTML, т.е. комментировать не надо
<head>
<title>Компьютерный супермаркет!</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251" />
</head>
<body bgcolor="#cccccc">
<center />
<table bgcolor="#bbccaa" border="0" cellspacing="0" cellpadding="0">
<tr><td><img src="/pic/lap.jpg" /></td>
<td><h1>    Магазин-Салон: Компьютеры и комплектущие    </h1></td></tr>
<tr>
<th bgcolor="#cccccc" colspan="2">
<?
echo '<div align="left">В корзине товаров - '.$count.'  <a href="http://localhost/view_basket.php">Содержимое</a></div></th>';
?>
</tr>
</table>
</html>
```

Исходный код файла myform4.php. Все пояснения приведены в комментариях исходного кода.

```
#здесь вся форма написана на html, т.е. комментарии не нужны
<html>
<p /><hr /><form method="post" action="lab4.php" enctype="multipart/form-data" name="main2">
<center />
<table bgcolor="#bbbbaa" border="5" width="60%">
<tr><th colspan="3"><h3>Выберите свой любимый компьютер</h3></th></tr>
<tr align="center">
<td><img src="/pic/comp1.jpg" /></td>
<td>Компьютер от фирмы <br>Hewlett Packard</td>
<td><input type="submit" name="hp" value="Выбрать" /></td>
</tr>
<tr align="center">
<td><img src="/pic/comp2.jpg" /></td>
<td>Компьютер от фирмы <br>Dell Computers</td>
<td><input type="submit" name="dell" value="Выбрать" /></td>
</tr>
<tr align="center">
<td><img src="/pic/comp3.jpg" /></td>
<td>Компьютер от фирмы <br>ASUS Computers</td>
<td><input type="submit" name="asus" value="Выбрать" /></td>
</tr>
</table>
</form>
</html
```

Исходный код файла end_out.php. Все пояснения приведены в комментариях исходного кода.

```
#здесь вся форма написана на html, т.е. комментарии не нужны
<html>
```

```
<hr />
<div align="left"><a href="http://localhost/lab4.php">На главную</a></div>
<div align="right">My Web Form</div>
</body>
</html>
```

Исходный код файла view_basket.php. Все пояснения приведены в комментариях исходного кода.

```
<?php                   #начало обычное
session_start();        #стартуем сессию
$count = count($_POST);     #если была нажата любая кнопка
if($count > 0) {
        if(isset($_POST['zakaz'])) {  #если была нажата кнопка оформления заказа
#здесь просто обнуляем корзину, но самостоятельно нужно дописать перенос товара в mysql
                unset($_SESSION['BASKET']);
        } else {        #если была нажата кнопка редактирования корзины
                foreach($_POST as $key => $val) { $del = $key-1; }
                array_splice($_SESSION[BASKET], $del, 1);       #удаляем помеченный товар
        }
}
require "top_out.php";          #рисуем шапку
#далее, если корзина не пуста, отображаем содержимое
if(isset($_SESSION['BASKET']) && count($_SESSION['BASKET']) > 0) {
echo '<html><p /><hr /><center />';
echo '<form method="post" action="view_basket.php" enctype="multipart/form-data">';
echo '<table bgcolor="#bbbbaa" border="10" width="60%">';
echo '<tr><th colspan="4"><em><h3>Содержимое корзины</em></th></tr>';
echo '<tr align="center"><td><h4>Товар</td> <td><h4>Название</td> <td><h4>Цена</td>
<td><h4>Редактировать</td></tr>';

$count = 0; $total_price = 0;
foreach ($_SESSION['BASKET'] as $tovar) {
        $count++;
        $total_price += $tovar['price'];
echo '<tr align="center"><td>'.$tovar['sname'].'</td> <td>'.$tovar['lname'].'</td>
<td>'.$tovar['price'].'</td> <td><input type="submit" name='.$count.' value="Удалить"
/></td></tr>';
}
echo '<tr><th colspan="4"><em><h3>Всего выбрано товаров '.$count.' на сумму
'.$total_price.'</em></th></tr>';
echo '<tr><th colspan="4"><input type="submit" name="zakaz" value="Оформить покупку"
/></th></tr>';
echo '</table></form></html>';
} else {        #иначе, если корзина опустела по нажатию «Оформить заказ»
        if(isset($_POST['zakaz'])) {
                echo '<p /><hr /><li><p>Покупка оформлена. Заказ будет доставлен в
ближайшее время.<br></li><hr />';
        } else {        #иначе, если корзина просто пустая
                echo '<p /><hr /><li><p>У Вас в корзине нет товаров<br></li><hr />';
        }
}
require "end_out.php";          #вызываем прорисовку нижней части
```

```
exit;
?>
```

Пробуем и отлаживаем работу страницы, набирая http://bm08.ogti.loc/lab4.php
Вместо bm08 необходимо поставить свое имя.

Самостоятельная работа. Необходимо написать web-страницу на PHP, предлагающей пользователю самому собрать компьютер из предлагаемых комплектующих компонентов. В зависимости от выбранных компонентов сообщается цена компьютера.

3 Методические указания по самостоятельной работе

Для успешного освоения курса «Б.1.В.ДВ.7.2 Разработка и реализация сетевых протоколов» необходима самостоятельная работа. В настоящее время актуальными становятся требования к личным качествам современного студента – умению самостоятельно пополнять и обновлять знания, вести самостоятельный поиск необходимого материала, быть творческой личностью.

Самостоятельную работу по освоению дисциплины обучающимся следует начинать с изучения содержания рабочей учебной программы дисциплины, цели и задач, структуры и содержания курса, основной и дополнительной литературы, рекомендованной для самостоятельной работы.

Самостоятельная учебная деятельность является необходимым условием успешного обучения. Многие профессиональные навыки, способность мыслить и обобщать, делать выводы и строить суждения, выступать и слушать других, – все это развивается в процессе самостоятельной работы студентов.

Самостоятельная работа по освоению дисциплины включает:

- самостоятельное изучение разделов;
- самоподготовку (проработку и повторение лекционного материала и материала учебников и учебных пособий);
- подготовку к лабораторным работам;
- подготовку к рубежному и итоговому контролю.

Самостоятельная учебная работа – условие успешного окончания высшего учебного заведения. Она является равноправной формой учебных занятий, наряду с лекциями, семинарами, экзаменами и зачетами, но реализуемая во внеаудиторное время.

Эффективность аудиторных занятий во многом зависит от того, насколько умело студенты организуют в ходе них свою самостоятельную учебную познавательную деятельность. Такая работа также способствует самообразованию и самовоспитанию, осуществляемому в интересах повышения профессиональных компетенций, общей эрудиции и формировании личностных качеств.

Самостоятельная работа реализуется:

1. непосредственно в процессе аудиторных занятий – на лекциях, лабораторных занятиях, при проведении рубежного контроля;
2. в контакте с преподавателем вне рамок расписания – на консультациях по учебным вопросам, при ликвидации задолженностей, при выполнении индивидуальных заданий;
3. в библиотеке, дома, в общежитии, на кафедре при выполнении студентом учебных задач.

В процессе проведения самостоятельной работы необходимо производить подбор литературных источников, научной периодической печати и т.д

4 Методические указания по итоговому контролю

Итоговый контроль знаний по дисциплине «Б.1.В.ДВ.7.2 Разработка и реализация сетевых протоколов» проводится в форме экзамена. Для подготовки к итоговому контролю знаний по дисциплине «Б.1.В.ДВ.7.2 Разработка и реализация сетевых протоколов» обучающиеся используют перечень вопросов, приведенный в фонде оценочных средств. Экзамен проводится в устной форме. В экзаменационный билет включен один теоретический вопрос. На подготовку студенту отводится 20-25 минут. На дифференцированном зачете ответы обучающегося оцениваются с учетом их полноты, правильности и аргументированности с учетом шкалы оценивания.

Оценка «отлично» выставляется студенту, если он глубоко и прочно усвоил программный материал, исчерпывающе, последовательно, четко и логически его излагает, умеет тесно увязывать теорию с практикой, свободно справляется с вопросами и другими видами применения знаний, причем не затрудняется с ответом при видоизменении заданий, использует в ответе профессиональные термины, правильно обосновывает принятое решение.

Оценка «хорошо» выставляется студенту, если он твердо знает материал, грамотно и по существу излагает его, не допуская существенных неточностей в ответе на вопрос, правильно применяет теоретические положения при решении практических вопросов, владеет необходимыми навыками и приемами их выполнения.

Оценка «удовлетворительно» выставляется студенту, если он имеет знания только основного материала, но не усвоил его деталей, допускает неточности, недостаточно правильные формулировки, нарушения логической последовательности в изложении программного материала.

Оценка «неудовлетворительно» выставляется студенту за отсутствие знаний по дисциплине, представления по вопрос, непонимание материала по дисциплине, наличие коммуникативных «барьеров» в общении, отсутствие ответа на предложенный вопрос.

5 Учебно-методическое обеспечение дисциплины

5.1 Основная литература

1. **Смелянский, Р. Л. Компьютерные сети** [Текст] : в 2 т. / Р. Л. Смелянский. - М. : Академия, 2011. - (Высшее профессиональное образование)
Т.1 : Системы передачи данных. - 304 с.. - ISBN 978-5-7695-7151-0
Т.2 : Сети ЭВМ. - 240 с.. - ISBN 978-5-7695-7153-4

5.2 Дополнительная литература

1. **Программное обеспечение сетей ЭВМ** [Текст] : методические указания к выполнению лабораторных работ / сост. В. Н. Муллабаев. - Орск : Издательство Орского гуманитарно-технологического института (филиала) ОГУ, 2014. - 71 с.

5.3 Периодические издания

1. Журнал «ИНФОРМАЦИОННЫЕ СИСТЕМЫ И ТЕХНОЛОГИИ».
2. Журнал «ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ И ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ».

3. Журнал «МИР ПК + DVD».
4. Журнал «ВЕСТНИК КОМПЬЮТЕРНЫХ И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ».
5. Журнал «ОТКРЫТЫЕ СИСТЕМЫ. СУБД».
6. Журнал «ЖУРНАЛ СЕТЕВЫХ РЕШЕНИЙ/ LAN».

5.4 Интернет-ресурсы

1. http://www.intuit.ru – некоммерческое частное образовательное учреждение дополнительного профессионального образования «Интернет - Университет Информационных Технологий»
2. http://www.kb.mista.ru – архив статей об информационных технологиях на принципах Wikipedia.org