# Основы работы со строковыми данными в PHP. Кодировка. Анализ текста.

Лабораторная работа № А-8.

## ЦЕЛЬ РАБОТЫ

Получение представлений об организации хранения текста в PHP, принципов работы с ним, особенностей обработки текстовой информации в различной кодировке.

## ПРОДОЛЖИТЕЛЬНОСТЬ

2 академических часа (1 занятие)

## РЕЗУЛЬТАТ РАБОТЫ

Размещенные на Веб-сервере и доступные по протоколу http два документа:

- *index.html* – статический HTML-файл с формой ввода текста;
- *result.php* – анализатор введенного текста.

## ДОПОЛНИТЕЛЬНЫЕ ТРАБОВАНИЯ К РАБОТЕ

Первый документ содержит статический *HTML*-код в кодировке *UTF*-8 и включает в себя форму с полем для ввода многострочного текста. Кроме того, он содержит кнопку "*Анализировать*", при нажатии которой всегда открывается документ *result.php*, содержащий:

1. исходный текст;
2. информацию о тексте;
3. кнопку «Другой анализ».

Исходный текст выделяется цветом и курсивом. В случае если текста нет – выводится надпись: "Нет текста для анализа". Информация о тексте включает в себя:

1. количество символов в тексте (включая пробелы);
2. количество букв;
3. количество строчных и заглавных букв по отдельности;
4. количество знаков препинания;
5. количество цифр;
6. количество слов;
7. количество вхождений каждого символа текста (без различия верхнего и нижнего регистров);
8. список всех слов в тексте и количество их вхождений, отсортированный по алфавиту.

Результат анализа оформляется в виде таблицы с границами между ячейками. Кнопка "*Другой анализ*" реализуется с помощью тега `<a>`, при нажатии на нее вновь загружается первый документ *index.html*.

Корректно анализироваться должен как английский, так и русский текст.

## РЕКОМЕНДАЦИИ К СТРУКТУРЕ ПРОГРАММЫ

Верстка документа *index.html* не представляется сложной задачей, поэтому остановимся на втором документе. Стоит отметить, что т.к. оба документа связаны, то они должны быть выполнены в одной кодировке *UTF*-8, иначе данные из первого документа будут неверно интерпретироваться во втором.

Листинг А-8. 1

```
if( isset($_POST['data']) && $_POST['data'] )   // если передан текст для анализа
{
     echo '<div class="src_text">'.$_POST['data'].'</div>';    // выводим текст
```

```
    test_it( $_POST['data'] );                  // анализируем текст
}
else        // если текста нет или он пустой
    echo '<div class="src_error">Нет текста для анализа</div>'; // выводим ошибку
```

Итак, второй документ содержит программный код PHP для анализа текста. Первое что необходимо сделать – это узнать доступен ли вообще этот текст для анализа. Если он не был передан, т.е. в документ не были переданы данные из формы, или же был передан пустой текст – анализ не производится, а выводится соответствующее сообщение.

Если же текст был передан, и он содержит символы – то он выводится в блоке `<div>`, что позволяет с помощью *CSS* оформить его согласно требованиям лабораторной работы. Затем вызывается функция `test_it()`, которая собственно и анализирует текст. Рассмотрим ее более подробно.

Листинг А-8. 2

```
function test_it( $text )
{
    // количество символов в тексте определяется функцией размера текста
    echo 'Количество символов: '.strlen($text).'<br>';
    // определяем ассоциированный массив с цифрами
    $cifra=array( '0'=>true, '1'=>true, '2'=>true, '3'=>true, '4'=>true,
                  '5'=>true, '6'=>true, '7'=>true, '8'=>true, '9'=>true );

    // вводим переменные для хранения информации о:
    $cifra_amount=0;    // количество цифр в тексте
    $word_amount=0;     // количество слов в тексте
    $word='';      // текущее слово
    $words=array();     // список всех слов

    for($i=0; $i<strlen($text); $i++)   // перебираем все символы текста
    {
        if( array_key_exists($text[$i], $cifra) )    // если встретилась цифра
            $cifra_amount++;                    // увеличиваем счетчик цифр
        // если в тексте встретился пробел или текст закончился
        if( $text[$i]==' ' || $i==strlen($text)-1 )
        {
            if( $word ) // если есть текущее слово
            {
                // если текущее слово сохранено в списке слов
                if( isset($words[$word]) )
                    $words[ $word ]++; // увеличиваем число его повторов
                else
                    $words[ $word ]=1; // первый повтор слова
            }
            $word=''; // сбрасываем текущее слово
        }
        else    // если слово продолжается
            $word.=$text[$i]; //добавляем в текущее слово новый символ
    }
    // выводим количество цифр в тексте
    echo 'Количество цифр: '.$cifra_amount.'<br>';
    // выводим количество слов в тексте
    echo 'Количество слов: '.count($words).'<br>';
}
```

Для измерения длины текста в символах используется стандартная функция `strlen()`. Подсчет отдельной группы символов немного сложнее – в качестве примера подсчитаем количество цифр в тексте. Для этого необходимо найти способ, с помощью которого можно однозначно определить, принадлежит ли символ группе или нет. Это можно сделать несколькими путями. Первый и самый простой – с помощью условного оператора: `if( $symb=='0' || $symb=='1' || … $symb=='9' ) { … }`. Если в группе немного символов – такая запись вполне уместна. Если же их много – то можно использовать и другой подход, реализованный в примере листинге А-8. 2.

В начале производится формирование ассоциированного массива, ключами которого является группа символов – в данном случае все цифры. Тогда, проверяя наличие в этом массиве элемента с индексом равным проверяемому символу, можно однозначно сказать принадлежит ли этот символ группе или нет. Поэтому, последовательно в цикле перебирая все символы текста и проверяя каждый из них на вхождение в указанном массиве в качестве ключа, мы можем сосчитать количество цифр в тексте (переменная `$cifra_amount`).

Далее функция анализирует количество слов в тексте. Для этого используется признак окончания слова: наличие пробела, знака препинания или конец текста. Исходя из этого можно описать следующий алгоритм разбиения текста на слова.

1. Текущее слово инициализируется пустой строкой.
2. В цикле для всех символов текста:
   2.1. если символ пробел, знак препинания или он последний в тексте, то текущее слово добавляется в список слов и сбрасывается в пустую строку;
   2.2. иначе к текущему слову добавляется текущий символ.

В качестве примера в рассмотренном коде в качестве признака окончания слова используется только пробел. Перед циклом создается переменная `$word` инициализированная пустой строкой и пустой массив `$words`, в котором будет хранится информация о количестве вхождений слов в текст.

Тогда, перебирая в цикле все символы текста, в переменную `$word` добавляются символы до тех пор, пока не встретился пробел (или текст не закончился). Это значит, что в переменной найдено какое-то слово и необходимо учесть его присутствие в тексте. Чтобы продолжить искать следующее слово необходимо опять присвоить переменной `$word` пустую строку.

Например, для текста "*Привет всем*" в переменную по очереди будут добавлены символы "*П*", "*р*", "*и*", "*в*", "*е*" и "*т*", сформировав в ней строку "*Привет*". Далее `$word` опять будет пустой строкой и в нее опять-таки по очереди будут добавлены символы "*в*", "*с*", "*е*" и "*м*", сформировав следующее слово "*всем*".

Учет количества вхождений осуществляется с помощью ассоциированного массива `$words`: при нахождении слова проверяется наличие в массиве элемента с индексом в виде искомого слова. Если такой элемент найден, то его значение увеличивается на единицу; если нет – то элемент добавляется в массив и его значение инициализируется единицей.

Например, для текста "*привет всем привет*" будут последовательно определены три слова: "*привет*", "*всем*" и "*привет*". Перед анализом массив `$words` пуст. После нахождения первого слова осуществляется проверка на наличие в нем элемента с индексом `'привет'`, которого в пустом массиве естественно нет. Поэтому в массив добавляется первый элемент: `$words['привет']=1`. (значение элемента означает количество вхождений слова в тексте; при первой встрече слова оно естественно равно единице) Второе слов также отсутствует в массиве, поэтому после его нахождения в массив также добавляется элемент `$words['всем']=1`. Но третье слово уже было сформировано ранее, т.е. в массиве уже присутствует элемент с ключом `'привет'`, а, следовательно, его значение просто увеличивается на единицу, становясь равным количеству вхождений слова в текст. Если когда-либо слово "*привет*" опять встретится в тексте – значение элемента будет снова увеличено на 1, т.е. будет определено что это слово встретилось три раза.

Таким образом в массиве `$words` в качестве ключей выступают все слова текста, а в качестве значений – количество их повторений в нем.

Самостоятельно доработайте функцию для подсчета и вывода количества знаков препинания, строчных букв, заглавных букв и всех букв аналогичным цифрам способом; учета знаков препинания как признака окончания слова; вывода количества слов; упорядочивание массива слов и вывод информации о количестве слов в тексте.

Последним не рассмотренным заданием лабораторной работы является подсчет количества повторений каждого символа в тексте. Решим эту задачу аналогичным подсчету слов способом, для чего напишем отдельную функцию.

Листинг А-8. 3

```
function test_symbs( $text )
{
	$symbs=array(); // массив символов текста
	$l_text=strtolower( $text ); // переводим текст в нижний регистр

	// последовательно перебираем все символы текста
	for($i=0; $i<strlen($l_text); $i++)
	{
		if( isset($symbs[$l_text[$i]]) ) // если символ есть в массиве
			$symbs[$l_text[$i]]++;	// увеличиваем счетчик повторов
		else				// иначе
			$symbs[$l_text[$i]]=1; // добавляем символ в массив
	}
	return $symbs; // возвращаем массив с числом вхождений символов в тексте
}
```

Т.к. по условиям лабораторной работы число вхождений символов считается без учета регистра, то текст перед анализом целиком переводится в нижний регистр. Тогда в тексте все символы верхнего регистра перейдут в нижний, а число их вхождения в тексте будет автоматически добавлено к числу вхождений соответствующих им символов в нижнем регистре.

Подсчет числа вхождений символов аналогичен подсчету числа слов в тексте, за исключением того, что нет необходимости вводить дополнительную переменную для формирования текущего символа: он всегда известен. Поэтому, массив `$symbs`, аналогичный массиву `$words`, строится путем перебора всех символов текста. Если символ уже есть в списке (есть элемент массива с таким индексом) – число его повторов увеличивается на 1. Если нет – добавляем в массив новый элемент с ключом из этого символа.

Самостоятельно добавите вызов функции `test_symbs()` в функцию `test_it()` и напишите *PHP* код для вывода этой информации.

### Кодировка и возможные затруднения в реализации лабораторной работы

Как известно при использовании кодировки *UTF*-8 кириллические символы занимают 2 байта, в то время как латинские – 1 байт. Из-за этого использование обычных функций для обработки строк будет работать некорректно – все они будут считать, что любой символ занимает 1 байт. Для исправления ситуации можно пойти двумя путями:

- использовать функции для *multybyte*-строк;
- перекодировать строки.

В первом случае функции автоматически учитывают сколько байт занимает символ и возвращают правильный результат. К сожалению, библиотека с ними не всегда подключена и корректно работает. И если подключение библиотеки легко проверить с помощью условного оператора (например: `if( function_exists('mb_strtolower') ) { ... }` ), то корректность работы проверяется только на практике.

Второй способ более простой. Он заключается в том, что перед анализом текста он принудительно перекодируется из кодировки *UTF*-8 в *CP*1251, где каждому символу всегда соответствует 1 байт и все стандартные функции прекрасно работают.

Листинг А-8. 4

```
// перед анализом перекодируем текст из UTF-8 в CP1251
test_it( iconv("utf-8", "cp1251",$_POST['data']) );

function test_it( $text )
{
	...

	// непосредственно перед выводом перекодируем строку обратно в UTF-8
	echo iconv("cp1251", "utf-8", $words[$key]);
```

```
        ...
}
```

Но в этом случае необходимо помнить, что сайт использует кодировку *UTF*-8, поэтому непосредственно перед выводом результата анализа все строки должны перекодироваться обратно из *CP*1251 в *UTF*-8 – иначе символы будут отображаться неправильно.

## СПРАВОЧНАЯ ИНФОРМАЦИЯ

| | |
|---|---|
| Сортировка ассоциированного массива по ключам | `ksort(); // по возрастанию значений ключей`<br>`krsort($arr); // по убыванию значений ключей`<br>Функции возвращают *true* в случае успешной сортировки и *false* в случае ошибки. |
| Проверка существования функции | `function_exists('function_name');`<br>Возвращает *true* в случае существования функции с указанным именем; *false* – в противном случае. |
| Перевод регистра для строки | `// преобразует строку в нижний регистр`<br>`$s2=strtolower($s1);`<br>`// преобразует строку в верхний регистр`<br>`$s3=strtoupper($s4);` |
| Определение длины строки | `strtolower($str); // возвращает длину строки` |
| Функции для multybyte-строк | Синтаксис функций совпадает со стандартными функциями по обработке строк – перед именем добавляется префикс "mb_". Например: `mb_strtolower()`. |
| Смена кодировки у строки | `iconv( $in_charset, $out_charset, $str );`<br>Функция возвращает результат перекодировки строки `$str` из кодировки `$in_charset` в `$out_charset`. |

## КОНТРОЛЬНЫЕ ВОПРОСЫ К ЛАБОРАТОРНОЙ РАБОТЕ

Для успешной защиты работы помимо соответствующего требованиям результата необходимо уверенно отвечать на нижеперечисленные и другие вопросы, а также на контрольные вопросы всех предыдущих лабораторных работ.

1. Какие стандартные функции сортировки массивов есть в PHP и как они работают?
2. Как проверить была ли определена функция?
3. Как изменить регистр строки:
4. Что такое функции для multybyte-строк и в чем их отличие от стандартных?
5. Что такое кодировка?
6. Какие кодировки Вы знаете? В чем их отличия? Преимущества и недостатки?
7. Как перекодировать строку в PHP из одной кодировки в другую?