Курсовая работа

НЕЙМИНГ. РАЗРАБОТКА ГЕНЕРАТОРА

# Оглавление

Введение

1. Теоретические основы нейминга.

1.1 Методология создания имени.

1.2 Алгоритм работы генератора названий

1.3 Анализ методов решения

1.3.1 Алгоритм разбиения слова на слоги

1.3.2 Алгоритм группировки слогов в слова

2. Реализация алгоритма создания имен

2.1 Разработка функции деления слова на слоги

2.2 Осуществление группировки слогов в слова

2.3 Анализ полученного генератора

Заключение

Список литературы

# Введение

Сегодня придумать хорошее название для новой торговой марки труднее, чем выбрать имя новорожденному ребенку. Ведь на полках небольшого российского супермаркета располагается около семи тысяч наименований товаров, а в гипермаркетах и того больше - их количество превосходит 50 тысяч. Но имя товара способно не только помочь в продвижении товара на рынке, но и «потопить» его среди более «удачных» конкурентов. Как же подобрать название новому товару? Ответом на этот вопрос послужило появление нейминга.

Чаще всего в отечественной маркетинговой литературе термин «naming» переводят с английского как «присвоение имен, разработка названий новых продуктов, товаров, услуг, торговых марок, компаний».[2]

В настоящее время любое рекламное агентство обязательно включает в перечень услуг нейминг. Это связано с ростом числа предпринимателей мелкой и средней руки в России. Естественно, огромный спрос вызывает довольно жесткую конкуренцию среди нейминговых агентств. Побеждает та компания, в которой подход к выполнению заказа более оригинален, а алгоритмы дают существенные результаты даже в тех областях, в которых количество используемых названий довольно-таки велико (пищевая промышленность, торговля и т.д.).

Таким образом, генератор названий может стать серьёзным помощником любого неймингового агентства.

*Цель работы:* разработка генератора названий.

Для ее достижения нужно решить *задачи* поиска алгоритма генерации и написания программы, реализующей его.

Для нахождения необходимой информации мы в основном обращались к сети Internet, т.к. именно там необходимость нейминга ощущается особенно остро. При написании кода также понадобилось обратиться к литературе по программированию на языке PHP.[6]

Данная работа содержит исследовательскую и практическую части. В первой даётся описание данного алгоритма разработки генератора с точки зрения той теории нейминга, которую я нашла в статье «29 способов имя образования». Во второй описывается процесс его реализации, а также проводится анализ недостатков и достоинств полученного продукта.

# 1. Теоретические основы нейминга.

##

## 1.1 Методология создания имени.

Создание имени происходит поэтапно. В качестве примера приведем одну из систем, разработанную компанией Nametrade. [4]

*Первый этап* - этап позиционирования. Для того чтобы правильно начать придумывание имени для товара, услуги или фирмы, нужно продумать позиционирование продукта. Необходимо изучить рынок, узнать больше о конкурентах и в самой фирме: изучить все, что может быть полезным при работе над именем. Здесь следует выделить ключевые слова (значения), которые должны определять характер товара, область его применения, особенные качества.

*Второй этап* - этап генерации. Из собранных ранее ключевых слов придумывается какое-либо, объединяющее их воедино. Здесь выделяют несколько видов слов:

1. Обычные слова

Плюсы:

* + перекликаются с родом занятий компании или сферой использования продукта: Sprint, Oracle, Time Manager.

Минусы:

* + невозможность полного переноса смысла при выходе на новый рынок;
	+ невозможность регистрации большинства слов.
1. Неологизмы. Так называют имена, которых не существует ни в одном языке мира. Sony, Teflon и Pepsi – все это неологизмы. Неологизмы имеют как положительные, так и отрицательные стороны.

Плюсы:

* + неологизмы не имеют «истории»;
	+ имя товара может дать название целой категории (Xerox, Kleenex, Pampers);
	+ такие слова идеальны для товаров, которые продаются по всему миру.

 Минусы:

* они ничего не значат.
1. Сложносокращенные слова. Такие имена, как ThinkPad, PanAm и Aquafresh являются сложносокращенными словами. Вы описываете продукт с помощью обычных слов и имеете право на регистрацию названия. Проблемы с переводом таких названий на языки различных народов мира возникают реже, чем у названий, составленных из обычных слов.
2. Аббревиатуры. BP и IBM это - British Petroleum и International Business Machines. Чтобы такое имя запомнили, нужно много усилий: его тяжело запомнить, и оно ничего не говорит потенциальному покупателю.

*Третий этап* - проверка. Имена и названия глубоко анализируются на соответствие различным критериям, выдвинутым рынком и заказчиком. Так же имена проверяются специалистами по фонетике и семантике. Еще одна проверка – на звучание.

*Четвертый этап*: оценка заказчиком. Отобранные на предыдущем этапе слова передаются на оценку заказчику. Поскольку заказчик “варится” в своем бизнесе, то он может оценить с ходу, какие слова подходят для рынка, а какие не очень. Поредевший список перемещается на экзекуции следующего этапа.

*Пятый этап* – тестирование. На данном этапе отобранные имена должны проходить тестирования на фокус-группах. Это может сделать только исследовательская компания. На этом же этапе проверяются юридические аспекты будущей легализации имен.

*Шестой этап*: окончательный. Список имен с сопроводительными документами в виде отчетов о фокус-группах и юридических справок ложится на стол заказчика, который, спросив совета исполнителей, принимает финальное решение.

Из приведенных здесь этапов нас более всего интересует этап генерации. Теоретически в процессе генерации человек должен являться непосредственным участником. Но, как мы видим, наиболее перспективными являются названия-неологизмы. Однако всем известно, как тяжело придумать абсолютно новое слово, которое должно удовлетворять всем критериям «хорошего» названия, а именно: [4]

1. Короткое;

2. Отличное от других и уникальное;

3. Связанное с реальностью;

4. Устанавливающее коммуникационный процесс;

5. Запоминающееся;

6. Благозвучное;

7. Без негативных ассоциаций.

Процесс генерации слов такого вида наиболее осуществим на компьютере и наименее – в голове человека (в силу того, что любое придуманное человеком слово будет так или иначе похоже на уже известные).

В виду всего вышеизложенного, стоит конкретизировать поставленную цель: разработать генератор названий-неологизмов, ассоциированных с некими ключевыми словами.

При удачной разработке мы должны получить набор ничего не значащих слов. В настоящее время некоторые нейминговые агентства могут похвастаться программами, определяющими эмоциональную окраску слова (как, например, программа ВААЛ, рекламируемая на naming.ru [3]), однако программы эти – коммерческая тайна, а разработка их в наших условиях невозможна, т.к. для этого необходим штат психологов. В связи с этим остальные этапы создания имени (т.е. все, кроме непосредственно генерации) должны осуществляться людьми.

## 1.2 Алгоритм работы генератора названий

На основе поставленной цели мы можем сформулировать задачу: составить алгоритм генерации неологизмов при наличии некого набора слов.

У нас есть несколько ключевых слов. Их количество и значение может быть произвольным. Новое слово должно базироваться на них. Первое, что может прийти в голову – составление новых слов из букв ключевых слов. На практике эта идея осуществляется легко. Но с таким, же успехом мы можем при помощи генератора случайных чисел выбрать несколько букв из алфавита (вероятность того, что эти буквы присутствуют в ключевых словах, очень велика), и из них «лепить» слова, варьируя положение букв внутри каждого из них. Во-первых, отсутствует даже подобие смысловой связи сгенерированных слов с предыдущими. Во-вторых, процесс генерации таких слов осложняется необходимостью постоянной слежки за морфологической и грамматической правильностью составляемого слова (как то: наличие в нем гласных, отсутствие несуществующих в русском языке сочетаний, вроде «хф», «чю» или «шь»). В-третьих, количество возможных комбинаций огромно, и нужны довольно-таки мощные ограничения на генерируемые слова.

Поэтому стоит задуматься над другими возможными путями, например, составления неологизмов из слогов ключевых слов. Этот путь более оправдан, т.к. каждый слог несет в себе какую-то эмоциональную окраску, вызывает ассоциации с тем или иным существующим в реальности словом. Единственная проблема заключается в подборе ключевых слов; в зависимости от их звучания определяется и звучание «новоиспеченных слов», известно, что многие вполне «мирные» слова в подсознании вызывают негативные эмоции. Идя по такому пути нужно правильно подбирать ключевые слова. Поскольку этим вопросом занимаются психологи, нас такой вариант генерации в принципе устраивает.

Еще более конкретная формулировка задачи: разработать генератор слов-неологизмов на базе слогов ключевых слов.

Эта задача автоматически разбивается на две подзадачи:

1. разбиение ключевых слов на слоги;
2. группировка слогов в слова.

В ходе решения этих подзадач стоит обратить внимание на то, что они выполняют противоположные операции, следовательно, стоит во время написания программы попытаться осуществлять некоторые простые функции в таком виде, чтобы они могли быть использованы как для решения первой подзадачи, так и для решения второй.

##

## 1.3 Анализ методов решения

###

### 1.3.1 Алгоритм разбиения слова на слоги

В русском языке существует вполне определенное правило разбиения слов на слоги. Однако в нем также много слов-исключений. Как правило, исключительной является ситуация, когда вопреки общему правилу буква не может быть отделена от слога, например, когда слог совпадает с какой-нибудь морфемой (чаще всего, корнем).

Для разрешения этой ситуации можно составить словарь, содержащий в себе все слова-исключения и их правильные разложения на слоги. Решение это несомненно правильное, но чересчур трудоёмкое и неблагодарное.

Более простой, но менее удобный в использовании способ – проверка разложения пользователем и в случае ошибочного разложения ввод правильного варианта с клавиатуры.

Поскольку основной целью курсовой работы является всё-таки разработка генератора, то такие недочеты программы можно опустить в связи с их непринципиальностью.

При разработке функции деления слова на слоги, удобно воспользоваться учебником русского языка и основными правилами деления на слоги:

1. Слова делятся на слоги: *го-ло-ва, ка-пу-ста(ка-пус-та),<…>, мо-я, я-ма* и др.
2. Слог может состоять из одного звука, из двух и более: *о-на, у-зел, ко-выль,<…>* и др.
3. В состав слога обязательно входит *гласный* звук, поэтому в слове столько слогов, сколько гласных звуков. <…>
4. Согласные звуки или начинают слог (*го-ло-ва*), или заканчивают его (*он, ум* и др.), или окружают гласный звук с обеих сторон (*шум, дым* и др.)[1].

###

### 1.3.2 Алгоритм группировки слогов в слова

Слоги можно группировать по нескольким признакам:

1. по принадлежности к тому или иному ключевому слову, например, не включать в новое слово более n слогов из одного ключевого слова (где n от 1 до количества слогов в слове).
2. по их положению в ключевых словах. Скажем, слоги в новых словах не могут идти в обратном порядке, чем в ключевых; или последние слоги ключевых слов не могут использоваться в качестве первых в новых, и т.д.
3. по повторяемости. Наиболее разумное ограничение – не допускать повторения слога в пределах одного нового слова.

По этим признакам можно вводить ограничения на постановку слога в слово. На данном этапе для нас это не главное. По этому мы рассмотрим общий случай, когда любой слог может оказаться на любом месте нового слова.

Стоит также задуматься о производительности данного алгоритма. Ведь чем больше ключевых слов будет введено, тем больше можно из них создать новых. Стоит учитывать также, что можно формировать слова с количеством слогов от 2-х до n, где n – общее число слогов. Этот фактор также увеличивает количество возможных вариантов.

Поэтому необходимо вводить какие-либо ограничения для выходных данных. Это может быть длина слов, количество слогов в них, степень повторяемости отдельных сочетаний (можно ввести в процентах от длины слова допустимое количество совпадения букв) или что-нибудь в этом роде.

При установке ограничений время выполнения программы значительно сократится. Реализация предложенных ограничений выглядит приблизительно одинаково, по этому для упрощения далее я буду ограничивать только количество слогов в слове, но стоит не забывать, что это не единственная возможность.

# 2. Реализация алгоритма создания имен

Реализацию полученного алгоритма я решила осуществить на языке PHP 4.0. Свой выбор я обосновываю его синтаксической близостью к языку программирования Си, с которым я наиболее знакома, и простотой работы с данными различных типов (что может особенно пригодиться для создания массивов строк). Кроме того, в процессе работы я смогу улучшить свои навыки работы с PHP, изучить его особенности.

Для ввода входных данных создадим с помощью языка разметки HTML страничку, содержащую форму. В её поля должны вводиться:

* количество ключевых слов (поле kol);
* непосредственно слова, записанные через запятую (поле slova);
* количество слогов в генерируемых словах (поле slog).

Данные этой формы будут обрабатываться файлом kurs.php, в результате работы которого будет генерироваться страничка со списком сгенерированных слов.

Сначала необходимо считать данные из формы и записать их в соответствующие переменные.

$kol=$\_POST["kol"];

$slova=$\_POST["slova"];

$slog=$\_POST["slog"];

В переменных kol и slog сейчас находятся какие-либо числа, а в переменной slova находится строка, состоящая из слов, разделенных запятыми и пробелами. Преобразуем её в массив строк (s):

$s=explode(",",$slo);

Для дальнейшей работы нам необходимо еще избавиться от ненужных нам пробельных символов (которые могут появиться при заполнении формы). Для этого воспользуемся встроенной функцией string trim (string str [, string charlist]). После такого рода обработки, можно будет приступить непосредственно к реализации алгоритма разбиения слова на слоги.

##

## 2.1 Разработка функции деления слова на слоги

Слова из массива строк s по одному будем посылать на обработку в функцию delslog($str).

**function delslog($str)**

{$k=k\_slog($str); //посчитаем количество слогов

$mas=slog($str, $k); //разобьём слово на слоги

$f=fopen("slogi.txt","a");//откроем файл slogi.txt

for($i=0;$i<count($mas);$i++) //для записи и

{//через «-» запишем в

$mas[$i].='-'; //него слоги

fwrite( $f,$mas[$i]);}

fclose($f);} //закроем файл

В этой части программы происходит вызов функций k\_slog($str)и slog($s, $k). Рассмотрим их подробнее.

**function k\_slog($str)** //считает количество слогов

{//бежим по строке

for($i=0, $q=0; $i < strlen($str); $i++)

if(glas($str[$i])) // если очередная буква

$q++; //гласная, увеличим счетчик

return $q; } //вернём значение счетчика

Функция slog($s, $k)выполняет непосредственно разбиение слова на слоги. Как видно из примеров в п. 1.3.1, многие слова можно разделить на слоги несколькими способами. По этому определить принадлежность каждой буквы к тому или иному слогу – довольно-таки сложная задача.

Разберемся сначала с гласными буквами. Каждая из них является основой слога, по этому достаточно только отслеживать, чтобы в слог не попала вторая гласная (в этом случае должен осуществиться переход к новому слогу). За это будет отвечать флажок q.

С согласными все сложнее. Из-за разночтений в правилах деления слов на слоги я решила придерживаться следующих пунктов:

1. Согласная и последующая гласная относятся к одному и тому же слогу;
2. Все согласные буквы, находящиеся перед первой гласной принадлежат одному слогу;
3. Все согласные буквы, расположенные после последней гласной принадлежат одному и тому же слогу;
4. Из нескольких согласных, расположенных в середине слова, первая относится к текущему слогу, остальные – к следующему (исключения составляют буквы «ъ» и «ь», которые «привязываются» к предыдущей согласной).

Последний пункт вызывает наибольшее количество сомнений относительно правильности, но вполне удовлетворяет изложенным в учебнике [1] правилам.

Получили функцию:

**function slog($s, $k)** //разбивает слово на слоги

{for($i=0, $q=0, $j=0; $i<strlen($s);)

{if(glas($s[$i]) && $q==0)//если очередная буква

{ //гласная и первая

$mas[$j].=$s[$i]; //припишем ее к слогу

$q=1; //установим флажок

$i++; //перейдем на след. букву}

else

if(!glas($s[$i]) && ($i==0 || $q!=1 || (glas($s[$i-1]) && !glas($s[$i+1])) || $j==$k-1))

/\*если буква не гласная и: первая, или до гласной, или между гласной и согласной, или последняя\*/

{

$mas[$j].=$s[$i];//припишем её к слогу

$i++; //перейдем на след. букву}

else

{ $q=0; //иначе снимем флажок и

$j++;} //перейдем на след. слог}

return $mas;} //вернём массив слогов

В приведенных функциях вызывается функция glas($a). Она выглядит так:

function glas($a) //проверяет, гласная ли буква 'а'

{

$gls='аеёиоуыэюя'; //Запишем строки из строчных и

$glb='АЕЁИОУЫЭЮЯ'; //заглавных букв

for($i=0; $i < 11; $i++)

if(substr\_count($gls,$a)>0|| //если 'а' входит

substr\_count($glb,$a) > 0)//в одну из строк

return 1; // вернём 1

return 0;}

Эта функция ищет вхождение переданной ей буквы a в двух строках, gls и glb, состоящих из строчных и заглавных гласных букв русского алфавита. Она возвращает 1, если буква присутствует в одной из строк (т.е. является гласной), иначе – 0.

Мы получили кусок основной программы, выполняющий первую задачу – разбиение слов на слоги.

В результате работы этой части программы файл slogi.txt заполняется слогами всех введенных слов, разделёнными знаком «-». Использование файла для временного хранения информации я считаю наиболее выгодным. Этот способ наиболее надежен и прост, потому что в противном случае пришлось бы либо вводить дополнительно переменные под хранение массивов слогов, либо наращивать один массив, но тогда могла бы возникнуть проблема с индексом, т.к. его надо было бы отправлять в качестве ещё одного параметра в функцию delslog($str). Использование нескольких функций вместо одной большой мне кажется также оправданным, т.к. это существенно упрощает восприятие текста программы. К тому же отдельные функции легче исправлять и дополнять.

##

## 2.2 Осуществление группировки слогов в слова

После разбиения всех введенных слов на слоги запускается функция группировки слогов в слова (group($k)). У неё только один входной параметр – количество слогов в новых словах.

В основе этой функции лежит алгоритм размещения с повторениями чисел 1..n в последовательности по k элементов.[5]

Рассмотрим эту функцию.

**function group($k)**

{

$f=fopen("slogi.txt","rt");//откроем файл slogi.txt

$a=explode("-",fgets($f));//запишем слоги в масс. а

fclose($f); //закроем файл

$n=count($a)-1; //посчитаем количество слогов

for($i=0;$i<$k;$i++){

 $x[$i]=1; //зададим начальную и

 $y[$i]=$n;} //конечную комбинацию чисел

while(prov($x,$y)) //пока они различны

{$p=$k; //пост. инд. в конец посл-ти

while($x[$p]==$n)//пока не найдем эл-т, отличный от

 $p--; //максимального, уменьш. р

$x[$p]++; //увелич р-й эл-т послед-ти

for($i=$p+1;$i<$k;$i++)

$x[$i]=1; //все след-ие делаем единицами

if(correct($x)) //если комбинация нам подходит

{for($i=0;$i<$k;$i++) //выводим слоги с соотв.

{ print $a[$x[$i]-1];}//номерами на экран

echo "<br>";} //переход на новую строку}

Эта функция вызывает prov($x, $y)и correct($x). Функция prov($x, $y)следит за совпадением начальной и конечной комбинациями. Это осуществляется методом простого перебора всех элементов массивов x и y. Если в результате количество совпадающих элементов равно количеству элементов в любом из массивов, то возвращается 0 (и алгоритм размещений с повторениями заканчивает свою работу).

**function prov($x, $y)**

{

$fl=0;

for($i=0;$i<count($y);$i++)

 if($x[$i]==$y[$i]) $fl++;

if($fl==count($y)) return 0;

return 1;}

Функция correct($x) проверяет полученную комбинацию на «пригодность». В данном случае, она отслеживает повторяющиеся слоги. Если они присутствуют, она не «пропускает» их к выводу на экран.

**function correct($x)**

{for($h=0;$h<count($x);$h++)

for($d=$h+1;$d<count($x);$d++)

if($x[$h]==$x[$d]) return 0;

return 1;}

В результате работы этой (завершительной) части программы на экран выводятся все слова длины, указанной в поле slog, не содержащие повторяющихся слогов.

##

## 2.3 Анализ полученного генератора

Полученная программа далеко неидеальна.

Основным недостатком её является трудоёмкость. Она довольно-таки велика за счет использования алгоритма размещения с повторениями, причем совершается явно лишняя работа, т.к. все размещения с повторениями как раз нам и не нужны - используем мы только слова без повторяющихся слогов. Ещё можно обратить внимание, что проверка букв в слове на принадлежность к гласным выполняется 2 раза, тогда как, заведя переменную, запоминающую места гласных букв в слове, мы могли бы прибегать к ней только один раз. Хотя по сравнению с масштабностью программы это не играет существенной роли, для красоты можно было бы это осуществить.

Вторым минусом программы я бы назвала отсутствие дополнительной обработки выводимых слов. Скажем, не осуществляется проверка слов на соответствие простейшим орфографическим правилам (в частности на наличие недопустимых сочетаний букв, которые могут возникнуть при «состыковке» слогов из разных слов). Не осуществляется и проверка на наличие одинаковых слогов в словах (что является ещё одной причиной появления слов с одинаковыми слогами, например, если ввести слово «мама»). Все проверки такого рода можно дописывать в функцию correct($x). Только тогда в качестве параметра ей следует передавать также массив слогов.

В генератор также можно было бы включить сортировку слогов в файле. Это упростило бы работу с выходными данными, т.к. они тоже были бы отсортированы. К тому же это можно было бы использовать для исключения из списка одинаковых слогов (о чем говорилось ранее).

Не смотря на недостатки, эта программа является рабочей и может являться если не готовым продуктом, то как минимум основой для его создания.

На мой взгляд, мне удалось написать программу довольно-таки просто, по максимуму используя возможности выбранного языка программирования.

# Заключение

Полученный в ходе данной работы генератор удовлетворяет большинству выдвинутых в процессе работы требований. Он действительно способен сгенерировать набор слов-неологизмов, основанных на ключевых словах. Далее из них мы должны самостоятельно выбирать те, которые удовлетворяют критериям «хорошего» названия (о которых писалось ранее). Не стоит забывать и о личных предпочтениях заказчиков. Этот процесс тоже довольно-таки трудоёмкий, но несомненно проще, чем придумывание этих слов.

*Варианты развития.* В качестве основных задач, неправленых на улучшение работы генератора можно выделить:

1. улучшение быстродействия;
2. углубление проверки выходных данных;
3. включение функции поиска наиболее благозвучных слов из полученных.

Однако уже в таком его состоянии его можно использовать в качестве помощника для составления названий. Эта программа, при должной доработке, может быть использована как нейминговыми агентствами, так и людьми, которым необходимо придумать название для чего угодно, но неохота платить за это деньги (например, когда речь идет о писателях-фантастах, которые уже месяц ломают голову, как назвать неизведанную планету…).

У нейминга в России, как и во сём мире, огромный потенциал для развития. Его необходимость современному обществу постоянно даёт толчок для возникновения принципиально новых и доработки существующих методов имяобразования. Компьютерное решение этой проблемы представляется на сегодняшний день одним из самых предпочтительных и удобных. Разработанный мной простейший генератор вряд ли внесет большой вклад в развитие этой отрасли, но сам факт решения этой задачи может послужить толчком к решению этой проблемы в более крупном масштабе, что со временем приведет к созданию чего-то более совершенного. В любом случае, в процессе разработки я на себе ощутила востребованность квалифицированного программиста в современной жизни.

# Список литературы

1. Бабайцева, В.В. «Русский язык: теория. 5 – 9кл.» Учебник для общеобразовательных учреждений. В.В. Бабайцева, Л.Д. Чеснокова – 9-е изд. – М.: Дрофа, 2000. – 48-49 с..
2. Бизнес-журнал [электронный ресурс]/ Джавед, Н. Случайный нейминг – Электрон. статья – 2005.- Режим доступа:
http://www.business-magazine.ru/oldbusiness/225706/. – Яз.рус.
3. Нейминогвое агентство Нейминг.Ру [Электронный ресурс]/ Лебедев-Любимов, А. Психологические проблемы нейминга, или как убеждать названием – Электрон. статья - 2006.- Режим доступа: http://www.naming.ru/mambo/index.php?option=com\_
content&task=view&id=12&Itemid=43
4. Patentoved.com [Электронный ресурс]/ Перция, В. 29 способов имяобразования – Электрон. статья – 2005. – Режим доступа: http://www.patentoved.com/content.php?id-32
5. Шень, А. Программирование: теоремы и задачи/ А. Шень. – М.: МЦНМО, 1995. – 43 – 44 с.
6. Котеров, Д.В. Самоучитель PHP 4. /Д.В. Котеров. — СПб.: БХВ-Петербург, 2001.