**Отчёт о проделанной работе**

**Библиотека функций для работы с числами произвольной точности**

В начале июня 2017 года я столкнулся с проблемой на языке PHP, которая вынудила меня разработать библиотеку для чисел с произвольной точностью.

На тот момент я выполнял задачки с факультатива по PHP. Чтобы пояснить, с какой проблемой я столкнулся, почему, и при каких обстоятельствах, я вставлю небольшой урывок из моего эссе для факультатива

Первой задачей, которую я решал, была задача №1 из блока №1.

Условие задачи:

*Величина Z содержит значение объема информации в байтах.*

*Написать программу, которая переводит значение Z в более крупные единицы измерения информации (Кбайт, Мбайт и т.д.).*

*Текущее значение величины Z нужно вводить в текстовое поле.*

Эту задачу я решал уже на VisualBasicScript, поэтому мне нужно было просто перевести код с VBS на PHP.

Суть программы в этом коде:

**Code:**

|  |
| --- |
| function convertMemory( $NumBytes )  {      if( abs( $NumBytes / pow( 1024, 8 ) ) > 0.8 )      {         return number\_format( $NumBytes / pow( 1024, 8 ), 2, ',', ' ' ) . ' ЙБ';      }      else if(abs($NumBytes/pow(1024, 7)) > 0.8)      {         return number\_format( $NumBytes / pow( 1024, 7 ), 2, ',', ' ' ) . ' ЗБ';      }      else if( abs( $NumBytes / pow( 1024, 6 ) ) > 0.8 )      {         return number\_format( $NumBytes / pow( 1024, 6 ), 2, ',', ' ' ) . ' ЭБ';      }      else if( abs( $NumBytes / pow( 1024, 5 ) ) > 0.8 )      {         return number\_format( $NumBytes / pow( 1024, 5 ), 2, ',', ' ' ) . ' ПБ';      }      else if( abs( $NumBytes / pow( 1024, 4 ) ) > 0.8 )      {         return number\_format( $NumBytes / pow( 1024, 4 ), 2, ',', ' ' ) . ' ТБ';      }      else if( abs( $NumBytes / pow( 1024, 3 ) ) > 0.8 )      {         return number\_format( $NumBytes / pow( 1024, 3 ), 2, ',', ' ' ) . ' ГБ';      }  ↶ ↶ ↶      else if( abs( $NumBytes / pow( 1024, 2 ) ) > 0.8 )      {         return number\_format( $NumBytes / pow( 1024, 2 ), 2, ',', ' ' ) . ' МБ';      }      else if( abs( $NumBytes / 1024 ) > 0.8 )      {         return number\_format( $NumBytes / 1024, 2,',',' ' ) . ' КБ';      }      else      {         return number\_format( $NumBytes, 0,',',' ' ) . ' Б';      }  } |

То есть введённое значение в байтах преобразовывается в как можно большую измерительную величину, округляя результирующее значение до второго знака после запятой, при этом каждые три разряда разделяются пробелом, а вещественная и целая части разделяются запятой.

Подобный подход используется в файловых системах (например, ОС Windows так делает в своём Explorer’е).

Но есть одно «но»: как я смог выяснить, функция *number\_format* использует функцию *sprintf* для разбиения разрядов символами-разделителями, которая, в свою очередь, имеет недостаток – она обрезает большие значения.

В ходе исследований, я выяснил, что проблема заключается в преобразовании самим PHP больших вещественных значений в экспоненциальный формат.

То есть число 999999999999999999991.9127312 будет преобразовано в 1.0Е+20.

Функция *sprintf* выводит число в обычном виде, а не научном, и для этого преобразует значение в обратном порядке. Совершенно элементарно, что она сдвигает число, которое находится перед символом Е (1.0 в нашем случае) влево или вправо (зависит от знака после Е) на то количество разрядов, которое стоит после Е (в нашем случае на +20, т.е. на 20 влево). По итогу число будет иметь вид 100000000000000000000.0, что, очевидно, отличается от изначального значения.

Мне это дело очень не понравилось и я начал бороться с этой проблемой.

Сначала я попробовал найти информацию о том, как можно отключить это нежелательное преобразование, которое производит интерпретатор, но это не увенчалось успехом: такой возможности попросту нет.

Затем я пришел к выводу, что нужно делать так, чтобы PHP не лез своими кривыми руками к числовым значениям, дабы не напортачить. Выход: использовать строковое представление числовых значений.

Я принялся переписывать функцию *number\_format*:

**Code:**

|  |
| --- |
| function str\_num\_format  (  $number,  $decimals = 2,  $dec\_point = ',',  $thousands\_sep = ' '  )  {  $array = preg\_split( "/[.]/", $number ); // Разделяем на 2 части (целую и  дробную)  while( $array[ 0 ][ 0 ] == '0' && strlen( $array[ 0 ] ) - 1 )  {  $array[ 0 ] = substr( $array[ 0 ], 1 ); // Убираем первый 0  }  $array[ 1 ] = substr( $array[ 1 ], 0, $decimals ); // Округляем (обрезаем,  если точнее) дробную часть  $array[ 0 ] = preg\_replace  (  '/\B(?=(?:\d{3})+(?!\d))/',  $thousands\_sep,  $array[ 0 ]  ); // Разделяем на разряды целую часть  $array[ 1 ] = preg\_replace  (  '/\B(?=(?:\d{3})+(?!\d))/',  $thousands\_sep,  $array[ 1 ]  ); // Разделяем на разряды дробную часть  return $array[0] . ( $array[1] ? $dec\_point . $array[1] : '' ); // Формируем  результирующую строку вида: <целая\_часть>[<разделитель><дробная\_часть>]  } |

Она получилась даже круче: в number\_format нельзя передавать только 1, и только 3 параметра. Обязательно нужно передавать или 2, или 4 параметра.

В моей же функции можно передавать 1, 2, 3 или 4 параметра.

Правда, моя функция уступает её двум аналогам по скорости выполнения.

Так вот, функция отлично работает, никого ни в чем не обрезает, но работает только для строковых значений.

Число на протяжении всех своих вычислений должно иметь строковое представление, дабы не потерять свою целостность. Для этого придётся переписать все математические функции под работу со строковыми числами, т.е. для произвольной точности. Такие библиотеки уже есть, но использовать я их не стал, т.к. хочу сам написать свою, извлечь из этого пользу и опыт. К тому же, лишнего проекта в портфолио не бывает.

Тему «Длинная арифметика» я взял на практику. Надеюсь, основные функции я за лето напишу.

Вот… Я начал думать: с чего начать и как действовать? Побывал на добром десятке интернет-страниц, которые были намерены помочь мне, но я бы не сказал, что у них это получилось: все методы, которые они мне предлагали, имели весомые для меня минусы. Позднее я подошел к двум преподавателям, чтобы попросить совет, как лучше всего организовать эту библиотеку (в основном, меня интересовал вопрос: в каком виде хранить входные и промежуточные данные?). Я изначально планировал хранить всё в виде строки, но это очень неоптимизированно получится. Тем не менее, я выбрал именно этот способ из множества других, ибо он прост, а времени у меня, как я на тот момент думал, было не так много.

В общем, «воду лить» я могу ещё очень долго, но мне уже не терпится рассказать о том, как я писал библиотеку, с какими проблемами столкнулся, как я их решил, и что с этого всего получилось.

Библиотеку я назвал LA (Long Arithmetic).

Она включает в себя два класса: LA и LAO (LA Operators). Первый класс используется для создания длинного числа и состоит из четырёх полей (целая часть, вещественная часть, разрядность вещественной части (наверное, не правильно выразился, поэтому далее приведу пример, который пояснит, что в этом поле хранится), отрицательность числа) и конструктора, внутри которого есть перегрузка для строк, вещественных и целых чисел.

Обещанный пример:

**Code:**

|  |
| --- |
| $la\_num = new LA( '9.01' ); // field1: '9'; field2: '1'; field3: '100'; field4: 'false'  $la\_num = new LA( '-0.07321e+2' ); // field1: '7'; field2: '321'; field3: '1000'; field4: 'true' |

То есть, второе и третье поле можно читать привычным нам образом: одна сотая, триста двадцать одна тысячная. Этот подход (имеется в виду хранение числа в таком виде) очень удобен для произведения некоторых математических операций.

Второй класс содержит в себе собственно математические функции, такие как суммирование (бинарный +), вычитание (бинарный -), инкремент (++), декремент (--), взятие по модулю, умножение (\*), а также операторы сравнения, такие как больше (>), меньше (<), больше или равно (>=), меньше или равно (<=), эквивалентность (==), не равно (!=), и логический оператор НЕ (!). В планах ещё добавить в этот список логический оператор ИЛИ ( || ), тернарный оператор, а также множество математических функций.

Оба класса при неожиданных входных данных выводят ошибки или предупреджения. Это можно будет увидеть в коде.

Далее я приложу код всея библиотеки, а под ним полный список с функциями и их описанием. В этом же описании приведу несколько проблем, с которыми мне пришлось столкнуться и напишу, как я с ними намерен бороться. Также было большое множество и других проблем, но о них я уж не буду писать.

Заранее извиняюсь за трудночитаемость кода на бумажных листах. Не знаю: поленился ли я, или просто пожалел несколько часов времени (возможно, и то, и другое), но выравнивать и красиво расставлять код на каждой страничке я не стал.

**Code:**

|  |
| --- |
| <?php      class LA // Long Arithmetic      {          // Class properties (class fields)          // Свойства класса (поля класса)          public $integer;          public $decimal;          public $decimal\_digit;          function \_\_construct( $number )          {              // In the code below we create the "constructor overload"              // В коде ниже мы создаём "перегрузку" конструктора              if( is\_string( $number ) )              {                  if( !is\_numeric( $number ) )                  {                      trigger\_error( '"Неверно задано значение для класса ' . \_\_CLASS\_\_ . '. Ожидается число."', E\_USER\_ERROR );                      exit;                  }              }              else if( is\_int( $number ) || is\_float( $number ) )              {                  trigger\_error( '"Возможна потеря данных. Ожидается строка."', E\_USER\_WARNING );                  $number = (string)$number;              }              else              {                  trigger\_error( '"Неверно задано значение для класса ' . \_\_CLASS\_\_ . '. Ожидается строка."', E\_USER\_ERROR );                  exit;              }              $this->negative = NULL;              if( is\_negative( $number ) )              {                  $this->negative = true;              }              $number = LAO::abs( $number );              if( is\_exp\_number( $number ) )              {                  $number = expToDecimal( $number );              }              else              {                  $number = removeLeadingZeros( $number );                  $number = removeZerosAfterDot( $number );              }              // Define the class fields              // Определяем поля класса              $this->integer = getIntPart( $number );              $this->decimal = getDecPart( $number );              $this->decimal\_digit = $this->decimal ? getDecDigit( $number ) : NULL;          }          function ToString( )          {              if( $this->integer === NULL )              {                  trigger\_error( '"Объект класса ' . \_\_CLASS\_\_ . ' не создан. Свойства отсутствуют. Возвращена пустая строка."', E\_USER\_WARNING );              }              return ( $this->negative && ( !!$this->integer || !!$this->decimal ) ? '-' : NULL ) . $this->integer . ( $this->decimal ? '.' : NULL ) . $this->decimal;          }      }  ?>    <?php      class LAO      {          public static function abs( $number )          {              if( is\_int( $number ) || is\_float( $number ) )              {                  trigger\_error( '"' . \_\_METHOD\_\_ . ': Возможна потеря данных. Ожидается строка."', E\_USER\_WARNING );                  $number = (string)$number;              }              else if ( is\_object( $number ) )              {                  if( get\_class( $number ) == 'LA' )                  {                      $number = $number->ToString( );                      if( is\_negative( $number ) || strpos( $number, '+' ) === 0 )                      {                          return new LA( substr( $number, 1 ) );                      }                      else                      {                          return new LA( $number );                      }                  }              }              if( is\_string( $number ) )              {                  if( is\_negative( $number ) || strpos( $number, '+' ) === 0 )                  {                      return substr( $number, 1 );                  }                  else                  {                      return $number;                  }              }              else              {                  trigger\_error( '"Неверно задано значение для метода ' . \_\_METHOD\_\_ . '. Ожидается строка или объект класса LA."', E\_USER\_ERROR );              }          }          public static function not( $number )          {              if( is\_string( $number ) )              {                  $number = new LA( $number );              }              if( is\_int( $number ) || is\_float( $number ) )              {                  trigger\_error( '"Возможна потеря данных. Ожидается строка. ' . \_\_METHOD\_\_ . '."', E\_USER\_WARNING );                  $number = (string)$number;              }              else if ( is\_object( $number ) )              {                  if( get\_class( $number ) == 'LA' )                  {                      $number = $number->ToString( );                  }              }              if( is\_string( $number ) )              {                  $number = removeLeadingZeros( $number );                  $number = removeZerosAfterDot( $number );                  return ( $number == '0' ? true : false );              }              else              {                  trigger\_error( '"Неверно задано значение для метода ' . \_\_METHOD\_\_ . '. Ожидается строка или объект класса LA."', E\_USER\_ERROR );              }          }          public static function eq( $one, $two )          {              // Перегрузка для первого параметра              if( is\_string( $one ) )              {                  $one = new LA( $one );              }              if( is\_int( $one ) || is\_float( $one ) )              {                  trigger\_error( '"' . \_\_METHOD\_\_ . ' [параметр 1]: Возможна потеря данных. Ожидается строка."', E\_USER\_WARNING );                  $one = (string)$one;              }              else if ( is\_object( $one ) )              {                  if( get\_class( $one ) == 'LA' )                  {                      $one = $one->ToString( );                  }              }              // Перегрузка для второго параметра              if( is\_string( $two ) )              {                  $two = new LA( $two );              }              if( is\_int( $two ) || is\_float( $two ) )              {                  trigger\_error( '"' . \_\_METHOD\_\_ . ' [параметр 2]: Возможна потеря данных. Ожидается строка."', E\_USER\_WARNING );                  $two = (string)$two;              }              else if ( is\_object( $two ) )              {                  if( get\_class( $two ) == 'LA' )                  {                      $two = $two->ToString( );                  }              }              if( is\_string( $one ) )              {                  if( is\_string( $two ) )                  {                      $one = removeLeadingZeros( $one );                      $one = removeZerosAfterDot( $one );                      $two = removeLeadingZeros( $two );                      $two = removeZerosAfterDot( $two );                      return ( $one == $two ? true : false );                  }                  else                  {                      trigger\_error( '"' . \_\_METHOD\_\_ . ' [параметр 2]: Неверно задано значение. Ожидается строка или объект класса LA."', E\_USER\_ERROR );                  }              }              else              {                  trigger\_error( '"' . \_\_METHOD\_\_ . ' [параметр 1]: Неверно задано значение. Ожидается строка или объект класса LA."', E\_USER\_ERROR );              }          }          public static function ne( $one, $two )          {              // Перегрузка для первого параметра              if( is\_string( $one ) )              {                  $one = new LA( $one );              }              if( is\_int( $one ) || is\_float( $one ) )              {                  trigger\_error( '"' . \_\_METHOD\_\_ . ' [параметр 1]: Возможна потеря данных. Ожидается строка."', E\_USER\_WARNING );                  $one = (string)$one;              }              else if ( is\_object( $one ) )              {                  if( get\_class( $one ) == 'LA' )                  {                      $one = $one->ToString( );                  }              }              // Перегрузка для второго параметра              if( is\_string( $two ) )              {                  $two = new LA( $two );              }              if( is\_int( $two ) || is\_float( $two ) )              {                  trigger\_error( '"' . \_\_METHOD\_\_ . ' [параметр 2]: Возможна потеря данных. Ожидается строка."', E\_USER\_WARNING );                  $two = (string)$two;              }              else if ( is\_object( $two ) )              {                  if( get\_class( $two ) == 'LA' )                  {                      $two= $two->ToString( );                  }              }              if( is\_string( $one ) )              {                  if( is\_string( $two ) )                  {                      $one = removeLeadingZeros( $one );                      $one = removeZerosAfterDot( $one );                      $two = removeLeadingZeros( $two );                      $two = removeZerosAfterDot( $two );                      return ( $one == $two ? false : true );                  }                  else                  {                      trigger\_error( '"' . \_\_METHOD\_\_ . ' [параметр 2]: Неверно задано значение. Ожидается строка или объект класса LA."', E\_USER\_ERROR );                  }              }              else              {                  trigger\_error( '"' . \_\_METHOD\_\_ . ' [параметр 1]: Неверно задано значение. Ожидается строка или объект класса LA."', E\_USER\_ERROR );              }              // Или же можно просто: return !( LAO::eq( $one, $two ) ), но тогда при некорректных параметрах ошибка будет выводиться из метода LAO::eq, что может немного затруднить нахождение ошибки.          }          public static function gt( $objOne, $objTwo )          {              if ( is\_object( $objOne ) )              {                  if( !( get\_class( $objOne ) == 'LA' ) )                  {                      $objOne = new LA( $objOne );                  }              }              else              {                  $objOne = new LA( $objOne );              }              if ( is\_object( $objTwo ) )              {                  if( !( get\_class( $objTwo ) == 'LA' ) )                  {                      $objTwo = new LA( $objTwo );                  }              }              else              {                  $objTwo = new LA( $objTwo );              }              if( $objOne->negative )              {                  if( !$objTwo->negative )                  {                      return false;                  }              }              else              {                  if( $objTwo->negative )                  {                      return true;                  }              }              $aOne = alignByDot              (                  str\_split                  (                       LAO::abs                       (                           $objOne->ToString( )                       )                  )              );              $aTwo = alignByDot              (                  str\_split                  (                       LAO::abs                       (                           $objTwo->ToString( )                       )                  )              );              $min\_key = getMinKey( array( $aOne, $aTwo ) );              $max\_key = getMaxKey( array( $aOne, $aTwo ) );              for( $i = $max\_key; $i >= $min\_key; $i-- )              {                  if( $aOne[ $i ] > $aTwo[ $i ] )                  {                      return true;                  }                  else if( $aTwo[ $i ] > $aOne[ $i ] )                  {                      return false;                  }              }              return false;          }          public static function lt( $objOne, $objTwo ) // less than          {              if ( is\_object( $objOne ) )              {                  if( !( get\_class( $objOne ) == 'LA' ) )                  {                      $objOne = new LA( $objOne );                  }              }              else              {                  $objOne = new LA( $objOne );              }              if ( is\_object( $objTwo ) )              {                  if( !( get\_class( $objTwo ) == 'LA' ) )                  {                      $objTwo = new LA( $objTwo );                  }              }              else              {                  $objTwo = new LA( $objTwo );              }              if( $objOne->negative )              {                  if( !$objTwo->negative )                  {                      return false;                  }              }              else              {                  if( $objTwo->negative )                  {                      return true;                  }              }              $aOne = alignByDot              (                  str\_split                  (                       LAO::abs                       (                           $objOne->ToString( )                       )                  )              );              $aTwo = alignByDot              (                  str\_split                  (                       LAO::abs                       (                           $objTwo->ToString( )                       )                  )              );              $min\_key = getMinKey( array( $aOne, $aTwo ) );              $max\_key = getMaxKey( array( $aOne, $aTwo ) );              for( $i = $max\_key; $i >= $min\_key; $i-- )              {                  if( $aOne[ $i ] < $aTwo[ $i ] )                  {                      return true;                  }                  else if( $aTwo[ $i ] < $aOne[ $i ] )                  {                      return false;                  }              }              return false;          }          public static function le( $objOne, $objTwo )// less than or equal          {              return LAO::eq( $objOne, $objTwo ) || LAO::lt( $objOne, $objTwo );          }          public static function ge( $objOne, $objTwo ) // greater than or equal          {              return LAO::eq( $objOne, $objTwo ) || LAO::lg( $objOne, $objTwo );          }          public static function sum( $objOne, $objTwo )          {              if ( is\_object( $objOne ) )              {                  if( !( get\_class( $objOne ) == 'LA' ) )                  {                      $objOne = new LA( $objOne );                  }              }              else              {                  $objOne = new LA( $objOne );              }              if ( is\_object( $objTwo ) )              {                  if( !( get\_class( $objTwo ) == 'LA' ) )                  {                      $objTwo = new LA( $objTwo );                  }              }              else              {                  $objTwo = new LA( $objTwo );              }              if( $objOne->negative )              {                  if( !$objTwo->negative )                  {                      return LAO::sub( $objTwo, LAO::abs( $objOne ) );                  }              }              else              {                  if( $objTwo->negative )                  {                      return LAO::sub( $objOne, LAO::abs( $objTwo ) );                  }              }              $aOne = alignByDot              (                  str\_split                  (                       LAO::abs                       (                           $objOne->ToString( )                       )                  )              );              $aTwo = alignByDot              (                  str\_split                  (                       LAO::abs                       (                           $objTwo->ToString( )                       )                  )              );              $inc\_next\_digit = false;              $min\_key = getMinKey( array( $aOne, $aTwo ) );              $max\_key = getMaxKey( array( $aOne, $aTwo ) );              for( $i = $min\_key; $i <= $max\_key + 1; $i++ )              {                  if( !$i )                  {                      $aThree[ $i ] = '.';                      continue;                  }                  $sum\_chars = $inc\_next\_digit + ( isset( $aOne[ $i ] ) ? $aOne[ $i ] : NULL ) + ( isset( $aTwo[ $i ] ) ? $aTwo[ $i ] : NULL );                  $inc\_next\_digit = false;                  if( $sum\_chars >= 10 )                  {                      $sum\_chars -= 10;                      $inc\_next\_digit = true;                  }                  $aThree[ $i ] = $sum\_chars;              }              $return\_object = new LA( implode( NULL, array\_reverse( $aThree ) ) );              $return\_object->negative = ( $objOne->negative && $objTwo->negative ) ? true : NULL;              return $return\_object;          }          public static function sub( $objOne, $objTwo )          {              if ( is\_object( $objOne ) )              {                  if( !( get\_class( $objOne ) == 'LA' ) )                  {                      $objOne = new LA( $objOne );                  }              }              else              {                  $objOne = new LA( $objOne );              }              if ( is\_object( $objTwo ) )              {                  if( !( get\_class( $objTwo ) == 'LA' ) )                  {                      $objTwo = new LA( $objTwo );                  }              }              else              {                  $objTwo = new LA( $objTwo );              }              if( $objOne->negative )              {                  if( !$objTwo->negative )                  {                      $return\_object = LAO::sum( LAO::abs( $objOne ), $objTwo );                      $return\_object->negative = true;                      return $return\_object;                  }                  else                  {                      return LAO::sub( LAO::abs( $objTwo ), $objOne );                  }              }              else              {                  if( $objTwo->negative )                  {                      return LAO::sum( $objOne, LAO::abs( $objTwo ) );                  }              }              if( $objTwo > $objOne )              {                  $return\_object = LAO::sub( LAO::abs( $objTwo ), LAO::abs( $objOne ) );                  $return\_object->negative = true;                  return $return\_object;              }              $aOne = alignByDot              (                  str\_split                  (                       LAO::abs                       (                           $objOne->ToString( )                       )                  )              );              $aTwo = alignByDot              (                  str\_split                  (                       LAO::abs                       (                           $objTwo->ToString( )                       )                  )              );              $dec\_next\_digit = false;              $min\_key = getMinKey( array( $aOne, $aTwo ) );              $max\_key = getMaxKey( array( $aOne, $aTwo ) );              for( $i = $min\_key; $i <= $max\_key + 1; $i++ )              {                  if( !$i )                  {                      $aThree[ $i ] = '.';                      continue;                  }                  $sub\_chars = ( isset( $aOne[ $i ] ) ? $aOne[ $i ] : NULL ) - $dec\_next\_digit - ( isset( $aTwo[ $i ] ) ? $aTwo[ $i ] : NULL );                  $dec\_next\_digit = false;                  if( $sub\_chars < 0 )                  {                      $sub\_chars += 10;                      $dec\_next\_digit = true;                  }                  $aThree[ $i ] = $sub\_chars;              }                return new LA( implode( NULL, array\_reverse( $aThree ) ) );          }          public static function inc( &$number, $digit = '0' )          {              if( is\_negative( $digit ) )              {                  $digit = LAO::abs( $digit );                  $inc\_number = shiftNumberToRight( '1', $digit );              }              else              {                  $inc\_number = shiftNumberToLeft( '1', $digit );              }              $number = LAO::sum( $number, $inc\_number);              return $number;          }          public static function dec( &$number, $digit = '0' )          {              if( is\_negative( $digit ) )              {                  $digit = LAO::abs( $digit );                  $dec\_number = shiftNumberToRight( '1', $digit );              }              else              {                  $dec\_number = shiftNumberToLeft( '1', $digit );              }              $number = LAO::sub( $number, $dec\_number);              return $number;          }          public static function mul( $objOne, $objTwo )          {              if ( is\_object( $objOne ) )              {                  if( !( get\_class( $objOne ) == 'LA' ) )                  {                      $objOne = new LA( $objOne );                  }              }              else              {                  $objOne = new LA( $objOne );              }              if ( is\_object( $objTwo ) )              {                  if( !( get\_class( $objTwo ) == 'LA' ) )                  {                      $objTwo = new LA( $objTwo );                  }              }              else              {                  $objTwo = new LA( $objTwo );              }              if( $objOne->negative ) // Первое число - отрицательное              {                  if( !$objTwo->negative ) // Первое число со знаком '-', второе - со знаком '+'                  {                      $return\_object = LAO::mul( LAO::abs( $objOne ), $objTwo );                      $return\_object->negative = true;                      return $return\_object;                  }                  else // - -                  {                      return LAO::mul( LAO::abs( $objOne ), LAO::abs( $objTwo ) );                  }              }              else              {                  if( $objTwo->negative ) // + -                  {                      $return\_object = LAO::mul( $objOne, LAO::abs( $objTwo ) );                      $return\_object->negative = true;                      return $return\_object;                  }              }              if( $objOne > $objTwo )              {                  return LAO::mul( $objTwo, $objOne );              }              if( $objOne->decimal || $objTwo->decimal )              {                  $digits = strlen( $objOne->decimal ) + strlen( $objTwo->decimal );                  return new LA                  (                      shiftNumberToRight                      (                          LAO::mul                          (                              new LA( $objOne->integer . $objOne->decimal ),                              new LA( $objTwo->integer . $objTwo->decimal )                          )->ToString(),                          $digits                      )                  );              }              $return\_object = new LA( '0' );              for( $i = $objOne; LAO::gt( $i, '0' ); LAO::dec( $i ) )              {                  $return\_object = LAO::sum( $return\_object, $objTwo );              }              return $return\_object;          }      }  ?>    <?php      function getMaxKey( $aArrays )      {          $max = 1;          foreach( $aArrays as $array )          {              foreach( $array as $key => $char )              {                  $max = ( $key > $max ? $key : $max );              }          }          return $max;      }      function getMinKey( $aArrays )      {          $min = 1;          foreach( $aArrays as $array )          {              foreach( $array as $key => $char )              {                  $min = ( $key < $min ? $key : $min );              }          }          return $min;      }      function alignByDot( $aNumber )      {          $dot\_position = getDotPosition( implode( NULL, $aNumber ) );          $dot\_position = $dot\_position === false ? strlen( implode( NULL, $aNumber ) ) : $dot\_position;          foreach( $aNumber as $key => $char )          {              $return\_array[ $dot\_position - $key ] = $char;          }          return $return\_array;      }      function is\_exp\_number( $number )      {          if( is\_numeric( $number ) && containi( $number, 'e' ) )          {              return true;          }          else          {              return false;          }      }      function getIntPart( $number )      {          return getStrBeforeStr( $number, '.' );      }      function getDecPart( $number )      {          return getStrAfterStr( $number, '.' );      }      function getDecDigit( $number )      {          return shiftNumberToLeft( '1', strlen( getDecPart( $number) ) );      }      function getShiftNumber( $number )      {          return getStrBeforeStri( $number, 'e' );      }      function getShiftDigit( $number )      {          return getStrAfterStri( $number, 'e' );      }      function expToDecimal( $number )      {          if( !is\_numeric( $number ) || !containi( $number, 'e' ) )          {              return false;          }          $minus = NULL;          if( is\_negative( $number ) )          {              $number = LAO::abs( $number );              $minus = '-';          }          $shift\_number = getShiftNumber( $number );          $shift\_digit = getShiftDigit( $number );          if( !LAO::abs( $shift\_digit ) )          {              $shift\_number = removeLeadingZeros( $shift\_number );              $shift\_number = removeZerosAfterDot( $shift\_number );              return $shift\_number;          }          if( is\_negative( $shift\_digit ) )          {              $shift\_digit = LAO::abs( $shift\_digit );              return $minus . shiftNumberToRight( $shift\_number, $shift\_digit );          }          else          {              return $minus . shiftNumberToLeft( $shift\_number, $shift\_digit );          }      }      function shiftNumberToRight( $number, $digit )      {          $number = removeLeadingZeros( $number );          $number\_without\_dot = str\_replace( '.', '', $number );          $old\_dot\_position = getDotPosition( $number );          if( $old\_dot\_position === false )          {              $old\_dot\_position = strlen( $number );          }          $zeros = NULL;          $new\_dot\_position = $old\_dot\_position - $digit;          if( $new\_dot\_position < 1 )          {              for( $i = $new\_dot\_position; $i < 0; $i++ )              {                  $zeros .= '0';              }              $new\_dot\_position -= leadingZerosCount( $number\_without\_dot );              $return\_number = removeLeadingZeros( $number\_without\_dot );              $return\_number = pasteStrToStr( $return\_number, '0.' . $zeros, 0 );              $return\_number = removeZerosAfterDot( $return\_number );              return $return\_number;          }          $new\_dot\_position -= leadingZerosCount( $number\_without\_dot );          $return\_number = removeLeadingZeros( $number\_without\_dot );          $return\_number = pasteStrToStr( $return\_number, '.', $new\_dot\_position );          $return\_number = removeZerosAfterDot( $return\_number );          return $return\_number;      }      function shiftNumberToLeft( $number, $digit )      {          $number\_without\_dot = str\_replace( '.', '', $number );          $old\_dot\_position = getDotPosition( $number );          if( $old\_dot\_position === false )          {              $old\_dot\_position = strlen( $number );          }          $zeros = NULL;          $new\_dot\_position = $old\_dot\_position + $digit;          if( $new\_dot\_position > strlen( $number\_without\_dot ) )          {              for( $i = ( $new\_dot\_position - $old\_dot\_position ); $i > 0; $i-- )              {                  $zeros .= '0';              }          }          $return\_number = pasteStrToStr( $number\_without\_dot, $zeros, strlen( $number\_without\_dot ) ); // Вставляем ноли справа ( 00123 -> 001230000)          $new\_dot\_position -= leadingZerosCount( $return\_number );          $return\_number = removeLeadingZeros( $return\_number ); // Удаляем ведущие ноли (001230000 -> 1230000)          while( $new\_dot\_position <= 0 )          {              $return\_number = pasteStrToStr( $return\_number, '0', 0);              $new\_dot\_position++;          }          $return\_number = pasteStrToStr( $return\_number, '.', $new\_dot\_position ); // Вставляем точку (1230000 -> 123.0000)          $return\_number = removeZerosAfterDot( $return\_number ); // Удаляем ноли после точки и точку, если они последняя в строке (123.0000 -> 123)          return $return\_number;      }      function removeLeadingZeros( $number )      {          while( $number[0] === '0' && ( isset( $number[1] ) ? $number[1] : NULL ) !==  '.' )          {              $number = substr( $number, 1);          }          return ( $number === false ? '0' : $number );      }      function leadingZerosCount( $number )      {          $i = 0;          $sum = 0;          while( $number[$i] === '0' || $number[$i] === '.' )          {              if( $number[$i] === '.' )              {                  continue;              }              $sum++;              $i++;          }          return $sum;      }      function removeZerosAfterDot( $number )      {          while( substr( $number, -1 ) === '0' || substr( $number, -1 ) === '.' )          {              if( substr( $number, -1 ) === '.' )              {                  return $number = substr( $number, 0, -1 );              }              if( getDotPosition( $number ) !== false )              {                  $number = substr( $number, 0, -1 );              }              else              {                  return $number;              }          }          $number = $number === '' ? '0' : $number;          return $number;      }      function pasteStrToStr( $train, $railcar, $position)      {        if( $position < 0 || $position > strlen( $train ) )        {          return false;        }        else        {          return substr( $train, 0, $position ) . $railcar . substr( $train, $position );        }      }      function getDotPosition( $number )      {          return strpos( $number, '.' );      }      function is\_negative( $number )      {          return ( substr( strval( $number ), 0, 1 ) == '-' );      }        function contain( $haystack, $needle )      {          return ( strpos( $haystack, $needle ) !== false ) ? true : false;      }      function containi( $haystack, $needle )      {          return ( stripos( $haystack, $needle ) !== false ) ? true : false;      }      function getStrBeforeStr( $str, $wall )      {         $wall\_position = strpos( $str, $wall );         return ( $wall\_position !== false ) ? substr( $str, 0, $wall\_position ) : $str;      }      function getStrAfterStr( $str, $wall )      {         $wall\_position = strpos( $str, $wall );         return ( $wall\_position !== false ) ? substr( $str, $wall\_position + 1 ) : NULL;      }      function getStrBeforeStri( $str, $wall )      {         $wall\_position = stripos( $str, $wall );         return ( $wall\_position !== false ) ? substr( $str, 0, $wall\_position ) : $str;      }      function getStrAfterStri( $str, $wall )      {         $wall\_position = stripos( $str, $wall );         return ( $wall\_position !== false ) ? substr( $str, $wall\_position + 1 ) : NULL;      }  ?> |

**Список с описанием функций (документация):**

1. class LA
   1. function \_\_construct( $number )

Конструктор класса LA. Принимает число. Если пришло не число или не

строка, то возвращает ошибку. Если пришло число типа int или float —

возвращает предупреждение о возможной потере данных. Создаёт четыре поля:

public $integer;

public $decimal;

public $decimal\_digit;

public $negative;

Последние три будут иметь тип NULL, если не будут иметь значения.

* 1. function ToString( )

Возвращает число в представлении строки вида

<целая\_часть>[<разделитель><вещественная\_часть>].

1. class LAO

Сноска: параметрами каждой функции предполагаются объекты класса LA, но

возможны и другие типы.

* 1. public static function abs( $number )

Функция взятия по модулю.

Принимает число, строку, или объект класса LA. Возвращает строку, если на

входе строка, в других случаях – объект.

* 1. public static function not( $number )

Принимает число, строку, или объект класса LA. Возвращает истину, если

число отрицательно, иначе — ложь.

* 1. public static function eq( $one, $two )

Принимает числа, строки, или объекты класса LA. Возвращает истину, если

числа эквивалентны, иначе — ложь.

* 1. public static function ne( $one, $two )

Принимает число, строку, или объект класса LA. Противоположность функции

LAO::eq().

* 1. public static function gt( $objOne, $objTwo )

Принимает числа, строки, или объекты класса LA. Возвращает истину, если

первое число больше второго, иначе — ложь.

* 1. public static function lt( $objOne, $objTwo )

Принимает числа, строки, или объекты класса LA. Возвращает истину, если

первое число меньше второго, иначе — ложь.

* 1. public static function le( $objOne, $objTwo )

Принимает числа, строки, или объекты класса LA. Противоположность функции

LAO::gt().

* 1. public static function ge( $objOne, $objTwo )

Принимает числа, строки, или объекты класса LA. Противоположность функции

LAO::lt()

* 1. public static function sum( $objOne, $objTwo )

Принимает числа, строки, или объекты класса LA. Возвращает сумму чисел.

* 1. public static function sub( $objOne, $objTwo )

Принимает числа, строки, или объекты класса LA. Возвращает разницу чисел.

* 1. public static function inc( &$number, $digit = '0' )

Инкремент. Принимает числа, строки, или объекты класса LA. Первый

параметр — ссылка на число, второй — разряд, который будет увеличиваться

на 1. Возвращает новое число.

Сама по себе функция вызывает функцию LAO::sum().

Варианты использования:

$la = new LA( "5" );

LAO::inc( $la ) // $la = 6

$res = LAO::inc( $la, "2" ) // $la = 106, res = 106

$la = new LA( "5.5" );

$res = LAO::inc( $la, 1 ) // $la = 6.5, res = 6.5

$la = new LA( "-3" );

$res = LAO::inc( $la, $la ) // $la = -2.999, res = -2.999

$la = new LA( "0" );

$res = LAO::inc( $la, "-1" ) // $la = 0.1, res = 0.1

* 1. public static function dec( &$number, $digit = '0' )

Декремент. Принимает числа, строки, или объекты класса LA. Принцип

действия тот же, что и у инкремента.

* 1. public static function mul( $objOne, $objTwo )

Произведение чисел. Принимает числа, строки, или объекты класса LA.

На данный момент произведение заменяется сложением. Число m

суммируется ровно n раз, где m — большее число, а n — меньшее.

То есть в случае с выражением (21233123.213123123123123 \* 3) суммирование

вызовется 3 раза, а вот в случае с выражением (9999999999 \* 9999999999) —

9999999999 раз. Поэтому я планирую дополнить функцию умножением

«в столбик». В таком случае суммирование будет вызываться ровно n \* m

раз, где n — количество цифр в первом числе, а m — во втором. Например,

вызовов суммирования будет уже не 9999999999, а 100. Но если выражение

будет таким: (1234567890.0987654321 \* 2), то здесь выиграет первый

способ, ибо он вызовет суммирование всего 2 раза, а «в столбик» — 40 раз.

Оптимальнее будет сделать сравнение количеств вызовов суммирования

обоими алгоритмами и использовать тот алгоритм, который будет работать

быстрее.

1. Вспомогательные функции
   1. function getMaxKey( $aArrays )

Принимает массив массивов. Возвращает максимальный индекс среди элементов

всех массивов.

* 1. function getMinKey( $aArrays )

Аналогична функции getMaxKey(), но возвращает минимальный индекс.

* 1. function alignByDot( $aNumber )

Принимает массив символов (в данном случае — цифр). Выравнивает массив

таким образом, чтобы центром (нулевым элементом) была точка. Слева от

точки — 1, а справа — -1. Возвращает новый массив.

Пример:

Элемент: 1| 2| 3| .| 4| 5| 6| 7 >>>>>>> 1| 2| 3| .| 4| 5| 6| 7

Индекс: 0| 1| 2| 3| 4| 5| 6| 7 >>>>>>> 3| 2| 1| 0|-1|-2|-3|-4

* 1. function is\_exp\_number( $number )

Принимает строку. Проверяет, является ли строка числом и содержит ли

символ 'e' или 'E'. Возвращает истину, если условия выполнились, иначе —

ложь.

* 1. function getIntPart( $number )

Принимает строку. Возвращает целую часть в виде строки.

* 1. function getDecPart( $number )

Принимает строку. Возвращает вещественную часть в виде строки.

* 1. function getDecDigit( $number )

Принимает строку. Возвращает разрядность вещественной части.

* 1. function getShiftNumber( $number )

Принимает строку в научном формате. Возвращает строку — сдвигаемое

число.

* 1. function getShiftDigit( $number )

Принимает строку в научном формате. Возвращает строку — количество

разрядов, на которое требуется сместить число.

* 1. function expToDecimal( $number )

Принимает строку в научном формате. Приводит её к десятичному виду

и возвращает результат в виде строки.

* 1. function shiftNumberToRight( $number, $digit )

Принимает числа в представлении строк. Сдвигает число $number вправо (→)

на количество разрядов, равное параметру $digit. Возвращает результат

в виде строки.

* 1. function shiftNumberToLeft( $number, $digit )

Принимает числа в представлении строк. Сдвигает число $number влево (←)

на количество разрядов, равное параметру $digit. Возвращает результат

в виде строки.

* 1. function removeLeadingZeros( $number )

Принимает строку. Удаляет ведущие нули. Возвращает результат в виде

строки.

* 1. function leadingZerosCount( $number )

Принимает строку. Возвращает количество ведущих нулей.

* 1. function removeZerosAfterDot( $number )

Принимает строку. Если число вещественное, удаляет ненужные нули справа.

Возвращает результирующую строку.

* 1. function pasteStrToStr( $train, $railcar, $position )

Вставляет строку в строку на определённую позицию.

Принимает строку, строку и число соответственно. Возвращает результат.

* 1. function getDotPosition( $number )

Принимает строку. Возвращает позицию точки в строке.

Если точки нет — вернёт позицию, следующую за последним символом.

* 1. function is\_negative( $number )

Принимает строку. Возвращает истину, если число отрицательно, иначе —

ложь.

* 1. function contain( $haystack, $needle )

Принимает строки. Возвращает позицию строки $needle в строке $haystack.

Если не находит строку в строке, возвращает ложь.

Аналогична функции contain(), но не чувствительна к регистру.

* 1. function getStrBeforeStr( $str, $wall )

Принимает строки. Возвращает строку до «стены». Если «стену» — вторую

строку — не находит, то возвращает первую строку полностью.

* 1. function getStrAfterStr( $str, $wall )

Принимает строки. Возвращает строку после «стены». Если «стену» — вторую

строку — не находит, то возвращает NULL.

* 1. function containi( $haystack, $needle )
  2. function getStrBeforeStri( $str, $wall )
  3. function getStrAfterStri( $str, $wall )

Эти функции аналогичны функциям выше, но они не чувствительны к регистру.

**Выводы:**

Во время написания этой библиотеки я подкрепил свои знания языка PHP, объектно-ориентированного программирования, освоил множество полезной и интересной информации, с которой мне пришлось столкнуться, ну, и, конечно же, вспомнил школьные уроки арифметики ☺.