Циклические алгоритмы. Условия в алгоритмах. Табулирование функций.

Лабораторная работа № А-2.

ЦЕЛЬ РАБОТЫ

Изучение структуры и синтаксиса языка PHP, получение базовых навыков построения программ, использования основных операторов. Закрепление знаний использования циклов и условных операторов, как основы построения алгоритмов.

ПРОДОЛЖИТЕЛЬНОСТЬ

4 академических часа (2 занятия)

РЕЗУЛЬТАТ РАБОТЫ

Размещенный на Веб-сервере и доступный по протоколу http документ (страница сайта) с результатами вычислений значений математической функции.

ДОПОЛНИТЕЛЬНЫЕ ТРАБОВАНИЯ К РАБОТЕ

Перед написанием программного кода PHP, необходимо подготовить статическую HTML-страницу, которая будет использоваться как шаблон для этой и следующих лабораторных работа. Шаблон должен удовлетворять следующим требованиям.

1. Использование стандарта HTML5 и CSS3. Прохождение страницей валидации сервисом "http://validator.w3.org/" без ошибок и предупреждений.
2. Наличие тегов "header", "main" и "footer".
3. В шапке должен размещаться логотип университета.
4. В названии страницы (TITLE) и в ее шапке должны быть указаны:
    A. ФИО выполнившего работу;
    B. группа;
    C. номер лабораторной работы и номер варианта (если он есть).
5. Подвал (footer) должен быть "приклеен" к низу экрана и не реагировать на скроллинг.
6. Цвета заливки и текста шапки и соответствующие цвета подвала должны отличаться от основного цвета страницы.
7. Стили прописаны в отдельном файле "styles.css".
8. Основной блок страницы должен быть светло-серого фона с шрифтом черного цвета.
9. Страница сайта должна корректно отображаться в любом современном браузере.

Не забудьте скопировать шаблон перед использованием в лабораторной работе – в дальнейшем он потребуется вам в начальном виде.

После создания шаблона и его проверки необходимо внедрить внутри тега "`<main> … </main>`" PHP-код, который бы:

1. в явном виде инициализировал числовые переменные, хранящие начальное значение аргумента функции, количество вычисляемых значений функции, шаг изменения значения аргумента. максимальное и минимальное значение функции, после которых вычисления останавливаются;
2. в явном виде инициализировал строковую переменную, хранящую тип формируемой верстки;
3. вычислял значения функции для соответствующего количества аргументов, начиная от начального значения аргумента с заданным выше шагом;
4. в подвале выводил название типа верстки.
5. в зависимости от заданного типа верстки ('A', 'B', 'C', 'D', 'E') выводил аргументы функции и ее значения в следующем виде.
    A. Простая верстка текстом, без таблиц и блоков. Выводятся строки (разделитель – тег "`<br>`") вида: "*f(x)=y*", где *x* – текущий аргумент, *y* – значении функции от этого аргумента (например, "*f(8)=15.8*").

B. Маркированный список. Каждая строка "*f(x)=y*" выводится как элемент маркированного списка (тег "<ul>" и "<li>").
C. Нумерованный список. Каждая строка "*f(x)=y*" выводится как элемент нумерованного списка (тег "<ol>" и "<li>").
D. Табличная верстка. Выводится таблица (границы ячеек одинарные, толщина 1*px*, черный цвет) в первой колонке которой размещается номер строки таблицы, во второй – значения аргументов, в третьей – значения функций от этих аргументов.
E. Блочная верстка. Каждая строка "*f(x)=y*" выводится внутри блока (тег " "), причем все блоки располагаются по горизонтали (соответствующий режим обтекания текстом), имеют красную рамку толщиной 2*px*, отступ друг от друга на 8*px*.

6. вычислял и выводил максимальное, минимальное, среднее арифметическое всех значений функции, а также их сумму.

ВАРИАНТЫ ЗАДАНИЯ

Задания лабораторной работы одинаковы для всех вариантов, за исключением вычисляемых функций. Номер задания соответствует номеру студента в списке группы (его необходимо уточнить у преподавателя перед началом выполнения лабораторной работы), для 1*x* и 2*x* номеров берется вариант №*x*.

1.	$f(x)=\begin{cases}10\cdot x-5, \text{при } x\le 10\\ (x+3)\cdot x^2, \text{при } x>10 \text{ и } x<20\\ \frac{3}{x-25}+2, \text{при } x\ge 20\end{cases}$
2.	$f(x)=\begin{cases}\frac{10+x}{x}, \text{при } x\le 10\\ (x/7)\cdot(x-2), \text{при } x>10 \text{ и } x<20\\ x\cdot 8+2, \text{при } x\ge 20\end{cases}$
3.	$f(x)=\begin{cases}3\cdot x^3+2, \text{при } x\le 10\\ 5\cdot x+7, \text{при } x>10 \text{ и } x<20\\ \frac{x}{22-x}-x, \text{при } x\ge 20\end{cases}$
4.	$f(x)=\begin{cases}(5-x)/(1-\frac{x}{5}), \text{при } x\le 10\\ x^2/4+7, \text{при } x>10 \text{ и } x<20\\ 2\cdot x-21, \text{при } x\ge 20\end{cases}$
5.	$f(x)=\begin{cases}3\cdot x+9, \text{при } x\le 10\\ \frac{x+3}{x^2-121}, \text{при } x>10 \text{ и } x<20\\ x^2\cdot 4-11, \text{при } x\ge 20\end{cases}$
6.	$f(x)=\begin{cases}x^2\cdot 0.33+4, \text{при } x\le 10\\ 18\cdot x-3, \text{при } x>10 \text{ и } x<20\\ \frac{1}{x*0.1-2}+3, \text{при } x\ge 20\end{cases}$
7.	$f(x)=\begin{cases}\frac{6}{x-5}\cdot x-5, \text{при } x\le 10\\ (x^2-1)\cdot x+7, \text{при } x>10 \text{ и } x<20\\ 4\cdot x+5, \text{при } x\ge 20\end{cases}$
8.	$f(x)=\begin{cases}7\cdot x+18, \text{при } x\le 10\\ \frac{(x-17)}{8-x\cdot 0.5}, \text{при } x>10 \text{ и } x<20\\ (x+4)\cdot(x-7), \text{при } x\ge 20\end{cases}$
9.	$f(x)=\begin{cases}x^2\cdot(x-2)+4, \text{при } x\le 10\\ 11\cdot x-55, \text{при } x>10 \text{ и } x<20\\ \frac{x-100}{100-x}-{}^{x}/_{10}+2, \text{при } x\ge 20\end{cases}$
10.	$f(x)=\begin{cases}\frac{3}{x}+\frac{x}{3}-5, \text{при } x\le 10\\ (x-7)\cdot(x/8), \text{при } x>10 \text{ и } x<20\\ 3\cdot x+2, \text{при } x\ge 20\end{cases}$

Все вычисления значения функции должны производиться по правилам математики с округлением конечного результата до 3-х знаков после запятой. В случае невозможности реализовать вычисление функций для заданной области определения, это должно быть обосновано студеном при защите. При попытке деления на ноль в качестве значения функции выводится строка "*error*".

РЕКОМЕНДАЦИИ К СТРУКТУРЕ ПРОГРАММЫ

Естественно-языковое описание алгоритма PHP-программы достаточно точно пересекается с заданием. Поэтому сразу приступим к разбору программы непосредственно на PHP.

Листинг А-2. 1

```
<?php
	$start_value = -10;		// начальное значение аргумента
	$encounting = 10000;		// количество вычисляемых значений
	$step = 2;			// шаг изменения аргумента
	$type = 'A';			// тип верстки

	$x=$start_value;		// текущее значение аргумента. равно начальному
```

```
        for( $i=0; $i < $encounting; $i++ ) // цикл с заданным количеством итераций
        {
            if( $x <= 10 )        // если аргумент меньше или равен 10
                 $f = 32*x / 21;       // вычисляем функцию
            else                  // иначе
            if( $x <20 )                // если аргумент меньше 20
                 $f = x*x/3 + 7/(x-4);           // вычисляем функцию
            else                        // иначе
                 $f = ( 1 / (x-11) )*2 + x;      // вычисляем функцию

            echo 'f('.$x.')='.$f.'<br>';    // выводим значение функции

            $x += $step;              // вычисляем следующий аргумент
        }
?>
```

Согласно требованиям лабораторной работы, в начале программы мы инициализируем четыре переменных, в которых будут храниться соответствующие значения – про ограничение диапазона значений пока забудем. При изменении этих переменных выводимые данные будут меняться. В следующей строке мы определяем текущее значение аргумента функции, которое берется из соответствующей переменной.

Основа программы – цикл со счетчиком, который совершает заданное число итераций. При этом с помощью условного оператора определяется по какой формуле вычисляется и сохраняется в переменной `$f` значение функции. Вывод осуществляется согласно типу верстки "A" – с построчной разбивкой данных. В конце каждой итерации цикла вычисляется новое значение аргумента: к его старому значению прибавляется шаг инкремент из переменной `$step`.

Рассмотренный скрипт содержит сразу несколько погрешностей, а именно:

- не учитывает тип верстки;
- не оптимален;
- содержит потенциальную ошибку при делении на ноль.

Последовательно улучшим программу так, чтобы она удовлетворяла требованиям лабораторной работы.

Листинг А-2. 2

```
<?php
        $x = -10;      // начальное значение аргумента
        $encounting = 10000;      // количество вычисляемых значений
        $step = 2;                // шаг изменения аргумента
        $type = 'A';              // тип верстки

        // цикл с заданным количеством итераций
        for( $i=0; $i < $encounting; $i++, $x+=$step )
        {
            if( $x <= 10 )        // если аргумент меньше или равен 10
                 $f = 32*x / 21;       // вычисляем функцию
            else                  // иначе
            if( $x <20 )                // если аргумент меньше 20
                 $f = x*x/3 + 7/(x-4);      // вычисляем функцию
            else                  // иначе
            {
                 if( x == 22 )              // если аргумент равен 22
                      $f= 'error';          // не вычисляем функцию
                 else                  // иначе
                      $f = ( 1 / (x-22) )*2 + x;      // вычисляем функцию
            }

            echo 'f('.$x.')='.$f.'<br>';    // выводим значение функции
        }
?>
```

Первое что мы сделали – убрали лишнюю переменную *$start_value*. Она используется только один раз как источник значения при инициализации переменной *$x* – гораздо проще сразу инициализировать *$x* значением -10. Кроме того, увеличение аргумента на каждой итерации цикла можно также перенести в оператор `for`.

Далее, там, где возможно деление на ноль, следует предусмотреть безопасную обработку этой ситуации. В примере это возможно при *x*=4 и *x*=22 – т.к. в программе присутствуют дроби, в знаменателях которых есть выражения (*x*-4) и (*x*-22). Для второго значения аргумента следует проверить это в соответствующей ветке программы: если аргумент принимает значение 22, то вычисления не выполняются, а функция принимает значение "*error*".

Листинг А-2. 3

```
<?php
       $x = -10;       // начальное значение аргумента
       $encounting = 10000;      // количество вычисляемых значений
       $step = 2;                // шаг изменения аргумента
       $type = 'A';              // тип верстки

       if( $type == 'B' )        // если тип верстки B
           echo '<ul>';          // начинаем список

       // цикл с заданным количеством итераций
       for( $i=0; $i < $encounting; $i++, $x+=$step )
       {
           if( $x <= 10 )        // если аргумент меньше или равен 10
                $f = 32*x / 21;       // вычисляем функцию
           else                  // иначе
           if( $x <20 )                // если аргумент меньше 20
                $f = x*x/3 + 7/(x-4);      // вычисляем функцию
           else                  // иначе
           {
                if( x == 22 )              // если аргумент равен 22
                     $f= 'error';          // не вычисляем функцию
                else                  // иначе
                     $f = ( 1 / (x-22) )*2 + x;      // вычисляем функцию
           }

              if( $type == 'A' )                    // если тип верстки A
              {
                 echo 'f('.$x.')='.$f;      // выводим аргумент и значение функции
                 if( $i < $encounting-1 )   // если это не последняя итерация цикла
                           echo '<br>';          // выводим знак перевода строки
              }
              else                                  // иначе
              if( $type == 'B' )                    // если тип верстки B
              {
                     // выводим данные как пункт списка
                     echo '<li>f('. $x.')='.$f.'</li>';
              }
       }

       if( $type == 'B' )        // если тип верстки B
           echo '</ul>';         // закрываем тег списка
?>
```

В листинге А-2.2 остался неизменным тип верстки – "A", что не позволяет считать задание лабораторной работы полностью выполненным. Поэтому программа несколько усложниться. В зависимости от типа верстки в теле цикла формируется разный HTML-код: либо строки разделяются символами перевода строки
, либо оформляются как элементы списка.

В первом случае мы дополнительно отслеживаем итерацию цикла – если она последняя, то выводить
 не нужно, т.к. строк далее просто нет. Это часто необходимо в программировании, особенно если разделение производится более сложными объектами и элементами дизайна. Признаком того, что итерация не последняя является то, то ее номер меньше общего количества итераций мину один. При этом можно пойти другим путем: выводить символ
 не в конце

строки, а в начале. Тогда необходимо проверять итерацию не на признак последней, а на признак первой – часто это гораздо проще, да и такой код выглядит компактнее и элегантнее.

Вывод строки с аргументом и значением при типе верстки "B" немного проще – строка просто заключается в соответствующий тег. Но этого недостаточно для построения валидного HTML-кода – для списка необходимо использовать тег "<ul>", который в цикле не выводится. Конечно, мы можем прямо в цикле, использовав условные операторы с проверкой на первую и последнюю итерации цикла, вывести этот тег. Но гораздо проще просто вынести вывод тега за пределы цикла, с соответствующей проверкой типа верстки.

Теперь добавим в программу ограничения минимального и максимального значений функции, при появлении которых вычисления останавливаются. Это можно сделать тремя путями: использовать цикл с предусловием, цикл с постусловием или оператор `break`. Рассмотрим все варианты.

Листинг А-2. 4

```
<?php
       $min_value=10;      // минимальное значение, останавливающее вычисления
       $max_value=20;      // максимальное значение, останавливающее вычисления

       ...                 // другой код PHP

       // цикл с заданным количеством итераций
       for( $i=0; $i < $encounting; $i++, $x+=$step )
       {
           ...                   // другой код PHP

           if( $f>=$max_value || $f<$min_value )      // если вышли за рамки диапазона
                break;                                // закончить цикл досрочно

       }
       ...                 // другой код PHP
?>
```

В начале скрипта во всех трех вариантах добавляются две искомые переменные `$min_value` и `$max_value`. Для краткости не будем включать в листинг уже разобранный код, заменив его многоточием. Единственным изменением программы, от варианта А-2.3 является наличие в конце условного оператора, который при превышении вычисленного значения функции `$f` значения из переменной `$max_value` или приуменьшении `$f` значения из `$min_value` выполняет оператор `break`, который прекращает работу цикла. Таким образом при достижении указанных условий вычисления прекращаются, что и требовалось по условиям лабораторной работы.

В отличии от цикла со счетчиком, цикл с предусловием не использует счетчик, а работает до тех пор, пока выполняется указанное условие. В листинге А-2.5 это условие интерпретируется так: выполнять цикл до тех пор, пока переменная счетчик итераций `$i` меньше заданного значения И значение функции в указанных пределах ИЛИ это первая итерация. Обратите внимание – операция "И" имеет приоритет перед операцией "ИЛИ", поэтому второе логическое выражение взято в скобки. Кроме того, признаком первой итерации является нулевое значение переменной `$i`, что соответствует логическому значению `false` – поэтому взято логическое отрицание переменной `$i` (т.е. если `$i` и всегда `true`, кроме нулевого значения, то `!$i` – всегда ложно, кроме нулевого значения).

Листинг А-2. 5

```
<?php

       ...                 // другой код PHP

       $i=0; $f=0;
       // цикл с предусловием
       while( $i<$encounting && ($f>=$max_value || $f<$min_value || !$i) )
       {
```

```
            ...                     // другой код PHP
               $i++; $x+=$step;         // принудительно увеличиваем значения счетчиков
        }
        ...                 // другой код PHP
?>
```

В цикле с предусловием удобно то, что условие полностью записывается в начале цикла и он может выполняться не фиксированное, а заранее неизвестное число итераций. Платой за это является необходимость явно указывать ПЕРЕД циклом начальные значения всех используемых в условии переменных и самостоятельно увеличивать значения необходимых переменных в его конце.

Цикл с постусловием работает аналогично, но условие проверяется не до, а после выполнения первой итерации. Т.е. такой цикл гарантированно выполнится хотя бы один раз, нет необходимости инициализировать все переменные из условия.

Листинг А-2. 6

```
<?php

        ...                 // другой код PHP

        $i=0;                       // начальное значение счетчика итераций
        do  // цикл с постусловием
        {
            ...                     // другой код PHP
               $i++; $x+=$step;         // принудительно увеличиваем значения счетчиков
        }
        while ($i<$encounting && ($f>=$max_value || $f<$min_value) );
        ...                 // другой код PHP
?>
```

В рамках лабораторной работы исследуйте и представьте все три варианта цикла. Выберете и обоснуйте наиболее оптимальный тип цикла для сформулированной задачи.

Представленная программа уже хорошо выполняет задание лабораторной работы. Но необходимо доработать ее так, чтобы она учитывала возможность остальных типов верстки. Для этого удобно, а в рамках этой работы необходимо, использовать конструкцию выбора. Это, а также другие требования задания необходимо сделать самостоятельно.

СПРАВОЧНАЯ ИНФОРМАЦИЯ

<table>
<tr><td>Цикл со счетчиком</td><td><code>for($i=0; $i<100; $i++) { ... }</code></td></tr>
<tr><td>Цикл с предусловием</td><td><code>$x=0; while ($x<10) { ...; $x++; }</code></td></tr>
<tr><td>Цикл с постусловием</td><td><code>$x = 0; do { ...; } while ($x++<10);</code></td></tr>
<tr><td>Оператор <code>break</code></td><td>Прекращает выполнение цикла</td></tr>
<tr><td>Оператор <code>continue</code></td><td>Прекращает выполнение текущей ИТЕРАЦИИ цикла</td></tr>
<tr><td>Конструкция выбора</td><td><pre>switch ($x) {
case 0:
     ...
     break;
case 1:
     ...
     break;
default:
     ...
}</pre></td></tr>
<tr><td>Округление с указанной точностью</td><td><code>round($x, $precison);</code><table><tr><td><code>round(3.4)</code></td><td>3</td></tr></table></td></tr>
</table>

`round(3.5)`	`4`
`round(3.6)`	`4`
`round(3.6, 0)`	`4`
`round(1.95583, 2)`	`1.96`
`round(1241757, -3)`	`1242000`
`round(5.045, 2)`	`5.05`
`round(5.055, 2)`	`5.06`

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. Что такое цикл?
2. Что такое условный оператор?
3. Какие виды циклов бывают?
4. Чем цикл с предусловием отличается от цикла с постусловием?
5. Чем отличаются операторы break и continue?
6. Можно ли решить задачу листингов А-2.4 – 6 четвертым путем? Если да, то каким?
7. Зачем в листинге А-2.5 проверяется условие первой итерации цикла?
8. Что такое итерация?
9. Какой тип данных определяет условие?
10. Что такое булева алгебра?
11. Какие операции булевой алгебры (логические операции) Вы знаете?
12. Как в PHP округлить значение до заданной точности?
13. Как в PHP преобразовать вещественный тип к целому?
14. Какие виды округления используются в PHP?
15. Что такое конструкция выбора?
16. Почему в примере А-2.2 и других не надо обрабатывать возможность деления на ноли при x=4?
17. Как изменить код А-2.3 так, чтобы выводить символ перевода строки "
" не в конце сформированной строки с данными, а в ее начале?