Составление задач для WebWork.

Перевод с английского: Шарипов Р. А.

1. Заголовки файлов.

Информация о задаче размещается внутри файла задачи в виде комментариев. Такие комментарии могут располагаться в любом месте файла. Но чаще всего её помещают в начало файла. Вот пример типичного заголовка файла

```
## DBsubject(Calculus - single variable)
## DBchapter(Limits and continuity)
## DBsection(Evaluating limits - factoring)
## Level(2)
## KEYWORDS('limits', 'factoring', 'rational function')
## TitleText1(Calculus)
## EditionText1(5e)
## AuthorText1(Stewart)
## Section1(2.9)
## Problem1(22)
## Author(Jeff Holt)
## Institution(UVA)
## Language(en)
```

Каждый информационный комментарий оформляется как тэг:

```
## Tagname(tag value)
```

Вначале после `##` даётся имя тэга. Затем в скобках — значение тэга. Тэг начинается с самого начала строки. В одной строке только один тэг.

Обязательными являются только три тэга: `DBsubject`, `DBchapter` и `DBsection`. В задаче также рекомендуется указывать уровень сложности (или рейтинг задачи)

```
## Level(2)
```

Выделяется шесть уровней сложности:

1. Задачи уровня сложности 1 для решения требуют лишь помнить определённый факт. Например, значение определённой функции в характерной точке ($\sin \pi$) или формулировку определения.
2. Задачи уровня сложности 2 предполагают, что студент должен продемонстрировать понимание определённого факта. Здесь предполагается умение применить умение применить простые стандартные методы, которые студент усвоил. Не требуется никаких рассуждений для выбора метода.
3. Здесь реализованы более сложные методы, такие как дифференцирование произведений и дифференцирование сложной функции одновременно или интегрирование заменой переменных и интегрирование по частям.
4. Применение прикладных алгоритмов, например, анализ функции на экстремумы.
5. Текстовые задачи.
6. Теоретическое применение определений и построение доказательств.

Тэги, которые могут присутствовать как опции.

- Ключевые слова. Тэг ключевых слов не должен содержать кавычек или апострофов. Например, если необходим тэг `L'Hopital's rule` (правило Липиталя), то надо написать `LHopitals rule`.
- `TitleText1`, `EditionText1`, `AuthorText1`, `Section1`, and `Problem1` – это просто ссылка на соответствующую задачу в бумажном учебнике.

- `Language`: код из ISO 639.1 указывает язык, на котором написан текст задачи (английский, французский, немецкий и т.д.).

- `Author` – указывает фамилию автора задачи.

- `Institution` – указывает организацию, где была составлена задача.

2. Общие рекомендации.

Для составления задач в WebWork используется язык PG, созданный Майклом Гейджем (Michael Gage), Арни Пайзером (Arnie Pizer) и их студентами в Рочестерском университете (University of Rochester). Этот язык объединяет возможности языка Perrl4 до объектной эпохи и современного парсера Perl. При составлении новых задач рекомендуется использовать новые возможности MathObjects и структур PGML.

MathObjects, созданный Давидом Сервоном (David Cervone) даёт большую объектную ориентированность. Например, объект Formula("x^2+(sin x)/x") как отображаться в виде строки TeX или в виде формулы для калькулятора. Для того, чтобы использовать MathObjects необходимо подключить модуль `MathObjects.pl в команде loadMacros()`:

```
loadMacros(
 "PGstandard.pl",
 "MathObjects.pl",
 "PGcourse.pl",
 );
```

Для создания объекта MathObjects есть много способов:

- При помощи команды-конструктора (напрмер `Real(3.5)` или `Complex("3+4i")`).
- При помощи функции `Compute()`.
- `При помощи встроенного метода одного объекта, создающего другой объект. Например, Formula("sin(x)")->eval(x => pi/2)`).
- Путем связывания существующих объектов математическими операторами, например, так `$x = Formula("x"); $f = $x**2 + 2*$x + 1`).

Примеры создания объектов

```
$a = Real(3.5);
$a = Compute(3.5);
$a = Compute("3.5");

$b = Complex(3, 4);
$b = Complex("3 + 4i");
$b = Compute("3 + 4i");

$v = Vector(4,5,8);
$v = Vector([4,5,8]);
```

```
$v = Vector("<4,5,8>");
$v = Compute("<4,5,8>");

$f = Formula("sin(x^2) + 6");
$f = Compute("sin(x^2) + 6");
```

Использование функции `Compute()` предпочтительно, ибо в этом случае вы задаёте, то, что ожидается от студента и объект учитывает это при проверке ответа. Если вы напишете `Vector("<cos(pi/6),sin(pi/6), pi/6>")->cmp`, то студент в качестве правильного ответа увидит `<0.866025,0.5,0.523599>`.

Команды

```
$a = Compute("x+3",x => 2);
$a = Compute("x^2+y^2",x => 1, y => 2);
```

равносильны команде `$a = Real(5)`.

3. Отображение объектов в тексте задачи.

Пусть задан объект типа формула по команде `$f = Formula("(x+1)/(x-1)");`. В каждой задаче есть отображаемая часть, которая заключена между тегами `BEGIN_TEXT` и `END_TEXT`. Обычно формулы более естественно изображать в формате `TeX`. Для этого надо использовать метод `TeX` объекта `$f`:

```
BEGIN_TEXT
  \[f(x) = \{$f->TeX\}\]
END_TEXT
```

Квадратные скобки со слэшем ограничивают вывод формул отдельной строкой. Круглые скобки со слэшем означают вывод формулы в строке. Фигурные скобки со слэшем выделяют `$f->TeX` в единый комплекс для отображения.

Можно не указывать метод `TeX` явно, если поменять контекст

```
Context()->texStrings;
BEGIN_TEXT
  \[f(x) = $f\]
END_TEXT
Context()->normalStrings;
```

После использования в формуле контекст надо вернуть в нормальное состояние, в котором текст формул используется при вычислениях.

4. Проверка ответов при помощи объектов.

Каждый объект MathObject имеет метод для проверки ответа. Этот метод называется `cmp()`. Если формула введена командой `$f = Formula("(x+1)/(x-1)");`, то проверка того, что студент ввёл ту же самую формулу, достигается командой

```
ANS($f->cmp);
```

У метода `cmp()` есть множество опций. Например

```
ANS(Real(sqrt(2))->cmp(tolerance => .0001));
ANS(Formula("sqrt(x-10)")->cmp(limits => [10,12]));
```

Подробную информацию о проверках ответов можно найти на сайте Answer Checkers

in MathObjects. Различные опции, которые допускает проверка ответов, можно найти на сайте Answer Checker Options.

5. Методы объектов MathObject.

Объекты MathObject являются объектами языка Perl. Доступ к их методам осуществляется при помощи специализатора ->. Вот формат использования различных методов для объектов MathObject

```
$mathObject->method;               # метод без аргументов
$mathObject->method($arg);         # метод с одним аргументом
$mathObject->method($arg1,$arg2); # метлд с двумя аргументами
```

Поля данных объекта также доступны при помощи специализатора ->:

```
$mathObject->{имя поля данных}
```

Имеется много методов и полей данных, которые являются общими для всех объектов MathObject. Их описание можно найти на сайте Common MathObject Methods и на сайте Common MathObject Properties.

6. Экспериментирование с объектами MathObject.

Для экспериментов с объектами MathObject создан специальный сайт on-line PG labs.

7. Пример кода простейшей задачи для WebWork.

Разберём код простейшей задачи для WebWork. Все начинается с заголовка

```
# DESCRIPTION
# Простая задача, в которой от студента требуется
# продифференцировать тригонометрическую функцию.
# Задача написана Гавином Ларозе (Gavin LaRose),
# <glarose(at)umich(dot)edu>
# ENDDESCRIPTION

## DBsubject('WeBWorK')
## DBchapter('Demos')
## DBsection('Problem')
## KEYWORDS('')
## TitleText1('')
## EditionText1('')
## AuthorText1('')
## Section1('')
## Problem1('')
## Author('Gavin LaRose')
## Institution('UMich')
```

Далее следует инициализационная часть кода, которая содержит загрузку макросов:

```
DOCUMENT();
loadMacros("PGstandard.pl","MathObjects.pl",);
```

После этого идёт содержательная часть задачи

```
Context("Numeric");
$a = random(2,9,1);
```

```
$trigFunc = Formula("sin($a x)");
$trigDeriv = $trigFunc->D();
```

Команда контекст определяет некоторые особенности того, как интерпретируются вводимые далее команды. Подробное разъяснение этой команды можно найти на сайте Context List. Далее идёт инициализация переменной `$a` случайным значением. Этим обеспечиваются индивидуальные числовые значения для каждого студента. В данном случае команда `random(2,9,1);` производит случайную величину между 2 и 9 с шагом 1. Подробнее см. на сайте PGrandom.

Команда `Formula`, как уже говорилось выше, создаёт объект MathObject. У этого объекта имеется метод дифференцирования `D()`. Применение этого метода создаёт другой объект MathObject типа формула. Величины `$trigFunc` и `$trigDeriv` – это переменные, указывающие на объекты типа формула.

```
TEXT(beginproblem());
Context()->texStrings;
BEGIN_TEXT
Найдите производную функции \(f(x) = $trigFunc\).
$PAR
\(\frac{df}{dx} = \) \{ ans_rule(35) \}
END_TEXT
Context()->normalStrings;
```

Команда `TEXT(beginproblem());` создаёт заголовок задачи. После этого изменяется контекст для того, чтобы выводить строки в формате TeX. Между командами `BEGIN_TEXT` и `END_TEXT` располагается текст задачи, выводимый студенту. Команда `$PAR` завершает параграф. Список подобных команд форматирования приведён на сайте Display Macros. После вывода текста контекст возвращается в стандартное состояние.

Команда `\{ ans_rule(35)` вставляет в текст форму ввода данных шириной 35 символов.

```
ANS($trigDeriv->cmp());
```

Команда `ANS` делает проверку ответа, обращаясь к методу проверки, заложенному в объекте типа формула. Связь между `ans_rule(35)` и `ANS();` остаётся за кадром.

```
Context()->texStrings;
SOLUTION(EV3(<<'END_SOLUTION'));
$PAR РЕШЕНИЕ $PAR
Мы находим производную, применяя правило дифференцирования сложной функ-
ции. Внутренняя функция – это \($a x\), поэтому её производная – это \
($a\), а внешняя функция – это \(\sin(x)\), её производная – это \
(\cos(x)\). Поэтому решение даётся формулой \[ \frac{d}{dx} $trigFunc =
$trigDeriv. \]
END_SOLUTION
Context()->normalStrings;
```

Все, что находится между `SOLUTION` и `END_SOLUTION` – это решение задачи. Оно демонстрируется студенту лишь после того, как решение набора задач завершено. Параметр `EV3(<<'END_SOLUTION')` указывает на версию 3 интерпретатора текста и задаёт метку `END_SOLUTION`, на которой интерпретируемый текст заканчивается.

```
ENDDOCUMENT();
```

Команда `ENDDOCUMENT();` - это последняя команда в файле задачи.

8. Вторая простейшая задача для WebWork.

Как и в первом случае задача начинается с заголовка, идентифицирующего её:

```
# DESCRIPTION
# Простая задача, иллюстрирующая
# три общих типов задач.
# Задача написана Гавином Ларозе (Gavin LaRose),
# <glarose(at)umich(dot)edu>
# ENDDESCRIPTION

## DBsubject('WeBWorK')
## DBchapter('Demos')
## DBsection('Problem')
## KEYWORDS('')
## TitleText1('')
## EditionText1('')
## AuthorText1('')
## Section1('')
## Problem1('')
## Author('Gavin LaRose')
## Institution('UMich')

DOCUMENT();
loadMacros("PGstandard.pl","MathObjects.pl","PGchoicemacros.pl",);
```

В этой задаче загружается на один макропакет больше: `PGchoicemacros.pl`.

```
Context("Numeric");

# ИНИЦИАЛИЗАЦИЯ ПЕРВОЙ ЧАСТИ ЗАДАЧИ

Context()->variables->add(t=>'Real');
$a = random(2,9,1);
$func = Formula("cos($a t)");
$funcDeriv = $func->D('t');
$m = $funcDeriv->eval(t=>2);
$y0 = $func->eval(t=>2);
$line = Formula("$m (t - 2) + $y0");
```

Важно заметить, что переменная `t` добавляется в контекст. Список возможных модификаций контекста содержится на сайте Context Modification

```
# ИНИЦИАЛИЗАЦИЯ ВТОРОЙ ЧАСТИ ЗАДАЧИ
$radio = new_multiple_choice();
$radio->qa("Эта задача является ", "легкой");
$radio->extra("очень легкой", "трудной");
$radio->makeLast("невозможной");
```

Во второй части создаётся новый объект типа кнопок выбора. Строка `$radio->qa("Эта задача является ", "легкой");` формирует вопрос и правильный выбор ответа на него. Следующая строка добавляет возможные неправильные ответы на вопрос. Последняя строка добавляет ответ, который будет изображаться студенту последним. Остальные ответы изображаются для студента в произвольном порядке.

```
# ИНИЦИАЛИЗАЦИЯ ТРЕТЬЕЙ ЧАСТИ ЗАДАЧИ
Context()->strings->add(Истина=>{},Ложь=>{});
$strAns = String('Истина');
```

Здесь происходит добавление строк возможных ответов в контекст. Затем в переменной `$strAns` формируется строка правильного ответа.

```
TEXT(beginproblem());
Context()->texStrings;
BEGIN_TEXT
${BBOLD}Часть 1$EBOLD
$BR
Найдите уравнение прямой, касательной к графику функции
\(f(t) = $func\) в точке \(t=2\).
$BR
\(y = \) \{ ans_rule(35) \} (ответ записать в виде функции от переменной \
(t\).)
```

Команды `${BBOLD}` и `$EBOLD` жирным шрифтом текст, заключённый между ними. Команда `$BR` выполняет переход на новую строку. Команда `$PAR` начинает новый абзац.

```
$PAR
${BBOLD}Часть 2$EBOLD
$BR
\{ $radio->print_q() \}
\{ $radio->print_a() \}
```

Приведённые выше команды отображают вопрос для кнопки выбора и ответы для этой кнопки.

```
$PAR
${BBOLD}Part 3$EBOLD
$BR
(Определите, Истина или Ложь следующее высказывание: ) Всем преподавателям
нравится WeBWorK.  \{ ans_rule(6) \}

END_TEXT
Context()->normalStrings;
```

После вывода текста задачи контекст возвращается к стандартной обработке строк.

```
ANS( $line->cmp() );
ANS( radio_cmp( $radio->correct_ans() ) );
ANS( $strAns->cmp() );
```

Поскольку задача состоит из трёх частей, проверка ответа выполняется тремя командами `ANS()`. Во всех трёх случаях вызывается метод `cmp()` соответствующего объекта.

```
Context()->texStrings;
SOLUTION(EV3(<<'END_SOLUTION'));
$PAR SOLUTION $PAR
${BBOLD}Часть 1$EBOLD
$BR
Производная функции \(f(t) = $func\) равна
\(f'(t) = $funcDeriv\), так что кгловой коэффициент наклона касательной
равен \(f'(2) = $m\). Далее при \(t = 2\) мы имеем \(f(2) = $y0\), так что
уравнение прямой имеет вид \(y = $line\).

$PAR
${BBOLD}Часть 2$EBOLD
$BR
Очевидно, что задача является лёгкой.
```

```
$PAR
${BBOLD} Часть 3$EBOLD
$BR
Тщательно проведённый опрос показал, что про одного преподавателя WeBWorK
100% опрошенных дувают, что данное утверждение верно.

END_SOLUTION
Context()->normalStrings;

ENDDOCUMENT();
```

Команда `ENDDOCUMENT();` - это последняя команда в файле задачи.

9. Третья простейшая задача для WebWork.

Как и во втором случае, задача начинается с заголовка, идентифицирующего её:

```
# DESCRIPTION
# Простая задача, которая иллюстрирует то, как
# динамически генерируемые графики вставляются в задачу
# Задача написана Гавином Ларозе (Gavin LaRose),
# <glarose(at)umich(dot)edu>
# ENDDESCRIPTION

## DBsubject('WeBWorK')
## DBchapter('Demos')
## DBsection('Problem')
## KEYWORDS('graphs')
## TitleText1('')
## EditionText1('')
## AuthorText1('')
## Section1('')
## Problem1('')
## Author('Gavin LaRose')
## Institution('UMich')
```

Имеется полный список тем `DBsubject`, глав `DBchapter` и разделов `DBsection` на сайте chaps-and-secs. Имеется также полный список ключевых слов `KEYWORDS` на сайте keywords.

```
DOCUMENT();
loadMacros("PGstandard.pl","MathObjects.pl","PGgraphmacros.pl",);
```

В дополнение к стандартным, приходится подгружать макрос `PGgraphmacros.pl`.

```
Context("Numeric");
$root1 = random(2,4,2);
$root2 = $root1 + random(2,4,2);
$func = Formula("-(1/8)(x-$root1)(x-$root2)");
```

В контексте Numeric парабола определяется своими пересечениями с осью x-ов.

```
$ymin = $func->eval(x=>-2);
$ymax = $func->eval(x=>($root1+$root2)/2);
```

Находятся значения функции в точке `x=-2` и в точке максимума (заметим, что парабола направлена ветвями вниз). Это определяет границы, в которых строится график и число линий сетки.

```
$xmin = -2;
```

```
$xmax = $root2 + 1;
$xgrid = $xmax + 2;
$ygrid = $ymax - int($ymin) + 2;
```

Далее инициализируется объект типа график по команде `init_graph`.

```
$graph = init_graph($xmin, int($ymin)-1, $xmax,
$ymax+1, axes=>[0,0], grid=>[$xgrid,$ygrid],
size=>[150,150]);
```

Параметр `axes` определяет точку пересечения осей координат. Параметр `size` определяет количество пикселей в картинке. Параметр `grid` определяет густоту линий сетки. Ещё один широко применяемый параметр – это `ticks`, он определяет густоту засечек на координатных осях.

```
plot_functions($graph, "$func for x in " .
"<$xmin,$xmax> using color:blue " .
"and weight:2");
```

Эта команда строит график функции на площадке, подготовленной предыдущей функцией `init_graph`. Параметры построения задаются строкой, которая в примере разбита на части и склеивается при помощи оператора точка. Интервал мог бы быть `<$xmin,$xmax>`. В этом случае в концевых точках была бы проставлена жирная точка и выколотая точка. Угловые скобки запрещают простановку точек на концах графика.

```
$yint = $func->eval(x=>0);
```

В этой строке вычисляется координата пересечения графика с осью у-ов.

```
TEXT(beginproblem());
Context()->texStrings;
BEGIN_TEXT
Рассмотрим график
$BR
$BCENTER
\{ image(insertGraph($graph), tex_size=>100,
    height=>150, width=>150,
    extra_html_tags=>'alt="график параболы ' .
    'ветвями вниз, пересекающей ось х-ов ' .
    'в точках ' . $root1 . ' и ' .
    $root2 . ', пересекающей ось у-ов в точке ' .
    $yint . '."' ) \}
$ECENTER

$PAR
Найдите уравнение параболы:
$BR
\( y = \) \{ ans_rule(35) \}

END_TEXT
Context()->normalStrings;
```

Это текстовая часть задачи с встроенным в неё графиком. Дополнительно даётся альтернативный текст, который выводится браузером, если вывод картинок в нём отключен.

```
ANS( $func->cmp() );

Context()->texStrings;
SOLUTION(EV3(<<'END_SOLUTION'));
$PAR РЕШЕНИЕ $PAR
```

```
Поскольку мы знаем, что это парабола и поскольку мы знаем
точки её пересечения с осью x-ов \(x = $root1\)
и \(x = $root2\), мы заключаем, что её уравнение должно иметь вид:
\[ y = a (x - $root1) (x - $root2) \]
для некоторой константы \(a\). Пересечение с осью y-ов
\(y = $yint\), оно происходит при
\(x = 0\), так что мы должны иметь
\(a($root1)($root2) = $yint\), а значит
\(a = -\frac{1}{8}\), и наша искомая функция
\[ y = $func. \]
END_SOLUTION
Context()->normalStrings;

ENDDOCUMENT();
```

В разделе РЕШЕНИЕ также выводится форматированный текст.

10. Первая простейшая задача для WebWork, реализованная по-старому без объектов MathObjects.

Как и в предыдущем случае, задача начинается с заголовка, идентифицирующего её:

```
# DESCRIPTION
# A simple sample problem that asks students to
# differentiate a trigonometric function.
# WeBWorK problem written by Gavin LaRose,
# <glarose(at)umich(dot)edu>
# ENDDESCRIPTION

## DBsubject('WeBWorK')
## DBchapter('Demos')
## DBsection('Problem')
## KEYWORDS('')
## TitleText1('')
## EditionText1('')
## AuthorText1('')
## Section1('')
## Problem1('')
## Author('Gavin LaRose')
## Institution('UMich')
```

Далее идёт фрагмент с загрузкой необходимых макросов

```
DOCUMENT();
loadMacros("PGstandard.pl","MathObjects.pl","PGcourse.pl",);
```

Далее определяются функция и её производная. И функция, и производная изображается в виде текстовых строк. Производная вычисляется вручную.

```
Context("Numeric");
$a = random(2,9,1);
$trigFuncTeX = "\sin($a x)";
$trigDeriv = "$a*cos($a*x)";
$trigDerivTeX = "$a \cos($a x)";
```

Далее идёт вывод текста задачи визуально

```
TEXT(&beginproblem);
Context()->texStrings;
BEGIN_TEXT
```

```
Найдите производную функции \(f(x) = $trigFuncTeX\).
$PAR
\(\frac{df}{dx} = \) \{ ans_rule(35) \}
END_TEXT
Context()->normalStrings;
```

Следующая команда осуществляет проверку ответа

```
ANS(fun_cmp($trigDeriv));
```

Заметим, что сравнение ответа с тем, что вводит студент, производится не при помощи метода `cmp()`, связанного с объектом, а при помощи процедуры сравнения `fun_cmp()`.

```
Context()->texStrings;
SOLUTION(EV3(<<'END_SOLUTION'));
$PAR РЕШЕНИЕ $PAR
Мы находим производную при помощи
Правила дифференцирования сложной функции.  Внутренняя функция \($a x\),
Её производная \($a\), а внешняя функция \(\sin(x)\), её производная
\(\cos(x)\).  Таким образом получается ответ
\[ \frac{d}{dx} $trigFuncTeX = $trigDerivTeX. \]
END_SOLUTION
Context()->normalStrings;

ENDDOCUMENT();
```

11. Задача с множественным ответом.

Рассмотрим пример задачи с множеством бланков для ответов. В этой задаче ответы с различных бланков исследуются для принятия решения о том, правильно ли решена задача. Начало стандартное:

```
DOCUMENT();
loadMacros("PGstandard.pl","MathObjects.pl","parserMultiAnswer.pl",);
TEXT(beginproblem());
```

Но есть загрузка модуля `parserMultiAnswer.pl`. Далее программируется сама задача:

```
$fac1 = Formula("(1 - x)");
$fac2 = Formula("(1 + x)");
```

Это задание объектов типа формула. Затем создаётся объект типа `MultiAnswer`:

```
$multians = MultiAnswer($fac1, $fac2)->with(singleResult=>0,
  checker=>sub {
      my ( $correct, $student, $self ) = @_;
      my ( $f1stu, $f2stu ) = @{$student};
      my ( $f1, $f2 ) = @{$correct};
      if ( ($f1 == $f1stu && $f2 == $f2stu) ||
           ($f1 == $f2stu && $f2 == $f1stu) ) {
          return [1,1];
      } else {
          if ($f1 == $f1stu || $f2 == $f1stu) {
              return [1,0];
          } elsif ($f1 == $f2stu || $f2 == $f2stu) {
              return [0,1];
          } else {
              return [0,0];
          }
      }
```

```
    }
);
```

Метод `with` объекта `MultiAnswer` задает значения параметров, отличающиеся от значений в контексте. Установка `singleResult=>0` означает, что результат проверки будет давать отдельные значения для каждого ответа, введённого студентом. Кроме того, устанавливается своя процедура проверки `checker=>sub{}`, отличная от заданной в контексте. Разберём эту процедуру. Процедура проверки может вызываться с произвольным числом аргументов. Значок `@_` указывает на список из её аргументов. Ключевое слово `my` в Perl служит для задания локальных переменных в блоке. Команда

```
my($correct,$student,$self)=@_;
```

создаёт локальную копию списка аргументов в теле процедуры `sub{}`. Аргумент `$student` сам является списком. Команда

```
my($f1stu,$f2stu)=@{$student};
```

создаёт локальную копию этого списка. При этом важно, что имя переменной `$student` взято в фигурные скобки. Иначе величина `@$student` давала бы число элементов в списке `$student`.

Аргумент тоже является списком. Команда

```
my($f1,$f2)=@{$correct};
```

создаёт локальную копию этого списка. Далее идут команды сравнения в виде

```
if (условие) {команды в случае выполнения условия}
else {команды в случае если условие нарушено}
```

либо в виде

```
if (условие) {команды в случае выполнения первого условия}
elsif (условие) {команды в случае выполнения второго условия}
else {команды в случае если оба условия нарушены}
```

Такие команды проверки в теле процедуры `sub{}` вложены друг в друга. Знак `||` является логическим ИЛИ. Логическое И давалось бы знаком `&&`. Логическое отрицание НЕ даётся восклицательным знаком `!`. В языке Perl имеется также условный логический оператор `?`, который, например, применяется так:

```
$access=($user eq 'Sharipov' ? ' Полный доступ ' : 'Ограниченный доступ ');
```

Далее идёт текст задачи. Он очень простой:

```
BEGIN_TEXT
Разложите на множители \(1-x^2 = \big( \)
\{$multians->ans_rule(10)\}
\( \big) \big( \)
\{$multians->ans_rule(10)\}
\( \big) \)
END_TEXT
```

В нём создаются два окошка ввода, где студент должен напечатать множители `1-x` и `1+x`. Но вся проблема в том, что он эти множители может напечатать в любом порядке. И оба варианта будут правильными. Поэтому потребовалась сложная процедура проверки. Задача завершается вызовом этой сложной процедуры проверки как метода в созданном

объекте `$multians`:

```
ANS( $multians->cmp() );
ENDDOCUMENT();
```

Обэект MultiAnswer имеет некоторые дополнительные опции. Это `allowBlankAnswers` и `checkTypes`. Кроме того, если выставлено значение `singleResult => 1` , то допускаются опции

- `separator` даёт разделитель, который студент вводит своими ответами. По умолчанию разделителем служит двоеточие. Но можно указать, например, строку `'and'`, записав `separator=>'and'`
- `tex_separator` даёт разделитель, который используется при предпросмотре ответов студентом.

Запись `allowBlankAnswers => 1` внутри `MultiAnswer->with()` даёт возможность оставлять поля ответов пустыми. Запись `checkTypes => 0` внутри `MultiAnswer->with()` отключает проверку совпадения типов при вводе ответов. Больше подробностей об использовании объекта можно найти по ссылке Tie several blanks to a single answer checker.

12. Многошаговаые задачи.

Здесь рассматривается пример многошаговой задачи, состоящей из нескольких частей, которые отображаются последовательно. Имеется два способа составления таких задач. Первый непосредственно обращается ответам, полученным на предыдущем шаге. Второй использует макрос *`compoundProblem.pl`*. Рассмотри первый способ.

```
DOCUMENT();
loadMacros("PGstandard.pl","MathObjects.pl","parserPopUp.pl",);
TEXT(beginproblem());
```

Начало, как видим, стандартное. Далее идёт следующий код:

```
$a = random(1,9,2);
$evenOdd = PopUp(["?","even","odd"],"odd");
$evenNum = Compute(2);
```

Переменная `$a` содержит случайное нечётное число в диапазоне от 1 до 9. Переменная `$evenOdd` определяется как объект с выбором из трёх вариантов. Строка `"odd"` задаётся как правильный ответ. Текст задачи выводится с переключением контекста отображения строк в стандарте TeX:

```
Context()->texStrings;
BEGIN_TEXT
${BBOLD}Часть первая из двух:$EBOLD
$PAR
Является ли число \( $a \) чётным (even) или нечётным (odd)?
\{ $evenOdd->menu() \}
END_TEXT
Context()->normalStrings;
```

Возможность выбора ответа даётся студенту посредством метода `menu()` объекта `PopUp`. Проверка правильности выбора, сделанного студентом, осуществляется при помощи метода `cmp()` того же объекта:

```
ANS($evenOdd->cmp());
```

В следующих командах мы видим более глубокое погружение в механизм функционирования метода `cmp()`:

```
$ans_hash1=$evenOdd->cmp()->evaluate($inputs_ref->{ANS_NUM_TO_NAME(1)});
```

По-видимому, `$evenOdd->cmp()` - это самостоятельный объект, который может использоваться и чаще всего используется в функции `ANS()`. Но этот объект может иметь и другое использование. У него есть метод `evaluate()`. Переменная `$inputs_ref`, по всей видимости, берётся из окружения (контекста). Это список, элементы которого адресуются по имени. Функция `ANS_NUM_TO_NAME(1)` преобразует номер ответа, введённого студентом, в его имя. По имени извлекается сам ответ из структуры `$inputs_ref`. Полученный ответ сам является списком, адресуемым по имени. Поле `{score}` этого списка содержит информацию о правильности ответа. В случае, если это поле содержит значение 1, то это сигнализирует о правильности ответа, проверенного предыдущей командой `ANS()`:

```
if ($ans_hash1->{score} == 1) {
Context()->texStrings;
BEGIN_TEXT
$BR
${BBOLD}Часть вторая из двух:$EBOLD
$PAR
Дайте пример чётного числа (even number),
\(n\): \(n = \) \{ $evenNum->ans_rule() \}
END_TEXT
Context()->normalStrings;
```

Дальше производится проверка чётности числа, введённого студентом:

```
ANS($evenNum->cmp(checker=>sub{
    my($cor,$stu,$ans) = @_;
    return 0 == ($stu % 2);}));
```

Для этого используется процедура `sub{}`, которая возвращает логическое значение. Это логическое значение вычисляется при помощи оператора `%`. Оператор `%` символизирует остаток от деления одного числа на другое.
После приведённых команд следует закрывающая фигурная скобка:

```
}
```

Она завершает условный оператор, начатый выше. Затем следует команда завершения файла задачи:

```
ENDDOCUMENT();
```

Отметим, что после получения правильного ответа на первый вопрос и его проверки вторая часть задачи выводится вместе с первой, что является недостатком данного кода.

Второй способ программирования многошаговой задачи в WebWork использует макрос *compoundProblem.pl*. Рассмотри этот способ.

```
DOCUMENT();
loadMacros("PGstandard.pl","MathObjects.pl","parserPopUp.pl",
           "compoundProblem.pl",);
TEXT(beginproblem());
```

Начало опять же, как видим, стандартное. Следующие команды дублируют те же команды из первого способа:

```
$a = random(1,9,2);
$evenOdd = PopUp(["?","even","odd"],"odd");
$evenNum = Compute(2);
```

А вот эта команда уже отличается. Она создаёт объект типа `compoundProblem`:

```
$cp = new compoundProblem(parts=>2,
                                weights=>[1,2],nextStyle=>'Forced');
```

Аргумент `parts` указывает число частей в задаче. Аргумент `weights` указывает веса, присвоенные каждой из частей.

Отображение первой части задачи совпадает с тем, что было при первом способе:

```
 Context()->texStrings;
BEGIN_TEXT
${BBOLD}Часть первая из двух:$EBOLD
$PAR
Является ли число \( $a \) чётным (even) или нечётным (odd)?
\{ $evenOdd->menu() \}
END_TEXT
Context()->normalStrings;
```

Проверка первой части задачи также стандартное и не отличается от того, как это делалось при первом способе программирования:

```
ANS($evenOdd->cmp());
```

Дальнейшее даётся следующими командами:

```
if ($cp->part > 1)
{Context()->texStrings;
BEGIN_TEXT
$BR
${BBOLD}Часть вторая из двух:$EBOLD
$PAR
Дайте пример чётного числа (even number),
\(n\): \(n = \) \{ $evenNum->ans_rule() \}
END_TEXT
Context()->normalStrings;

  ANS( $evenNum->cmp( checker=>sub {
      my ( $cor, $stu, $ans ) = @_;
      return 0 == ($stu % 2); } ) );}
```

Здесь используется объект `$cp` типа `compoundProblem` и его поле `part`, которое должно быть больше единицы. Концовка задачи стандартная:

```
ENDDOCUMENT();
```

Объект типа `compoundProblem` может иметь целый ряд параметров. Вот их описание:

- `parts=>n` число частей в задаче;
- `weights=> [n1,…,ns]` относительные веса отдельных частей. Например, если задано `weights=>[2,1,1]`, то на первую часть придётся 50% от оценки за задачу, а на две оставшиеся, по 25%;
- `totalAnswers=>n` общее число полей для ввода ответа. Этот параметр используется при подсчёте относительных весов частей задачи, если они не указаны явно;
- `saveAllAnswers => 0` или `1`. Обычно содержание бланков ввода из предыдущих

частей задачи делаются доступными в последующих частях через переменные с именами, совпадающими с именами бланков. Установка делает ВСЕ бланки ответов доступными через переменные вида `$AnSwEr1`, `$AnSwEr2` и т. д. Если параметр не установлен `saveAllAnswers` явно, его значение выставляется равным нулю;

- `parserValues => 0` или `1`. Этот параметр определяет то, будут ли ответы из предыдущих частей передаваться в виде объектов или в виде строк, как их напечатал студент в бланках ответов;
- `nextVisible => значение`. Этот параметр определяет то, будет ли переход к следующей части задачи доступен студенту из предыдущей части задачи. Возможные значения `ifCorrect` (переход доступен если все предыдущие ответы верны), `Always` (переход доступен всегда), `Never` (переход происходит сам собой).
- `nextStyle => значение`. Этот параметр определяет то, как будет отображаться указатель перехода к следующей части задачи. Возможные значения включают `CheckBox`, `Button`, `Forced` (переход происходит сразу после введения ответа к предыдущей части) и `HTML` (переход можно оформить через HTML код).

13. Задача с матрицами в условии и в ответе.

Здесь рассматривается пример задачи с матрицей в качестве ответа. Заголовок задачи стандартен. Но подключен макрос `AnswerFormatHelp.pl`:

```
DOCUMENT();
loadMacros("PGstandard.pl","MathObjects.pl","AnswerFormatHelp.pl",);
```

В процессе инициализации объектов задачи используется матричный контекст:

```
TEXT(beginproblem());
Context("Matrix");

$A = Matrix([[random(-5,5,1),random(-5,5,1),random(-5,5,1)],
[random(-5,5,1),random(-5,5,1),random(-5,5,1)],]);

$B = Matrix([random(-5,5,1),random(-5,5,1),random(-5,5,1)]);

$answer = $A * ($B->transpose);
```

Переменные `$A` и `$B` инициализируются как объекты типа матрица. В объектах типа матрица уже заложен методы транспонирования. А перемножение матриц, по всей видимости, выполняется матричным контекстом.

Для отображения текста задачи студенту контекст меняется на `texStrings`:

```
Context()->texStrings;
BEGIN_TEXT
Положим
\[A = $A \ \ \mbox{и} \ \ B = $B.\]
Вычислите следующее произведение матриц:
$BR
$BR
\( A B^T = \)
\{ $answer->ans_array(5) \}
\{ AnswerFormatHelp("matrices") \}
END_TEXT
Context()->normalStrings;
```

Матричные объекты `$A` и `$B` умеют отображать себя в контексте `texStrings`. Матричный объект `$answer` умеет отображать ячейки для введения ответа через свой метод

`ans_array(5)`. Все ячейки имеют одинаковую ширину 5. Команда `AnswerFormatHelp("matrices")` отображает ссылку с подсказкой для студента.

Проверка ответа выглядит стандартно через метод `cmp()`:

```
$showPartialCorrectAnswers = 1;
ANS($answer->cmp());
```

Это потому, что матричный объект `$answer` имеет свой встроенный механизм проверки в методе `cmp()`. Настройка `$showPartialCorrectAnswers = 1;` должна позволять студенту видеть, какие из элементов матрицы посчитаны им правильно, а какие нет. Но при отработке в песочнице эта опция не срабатывает.

```
Context()->texStrings;
SOLUTION(EV3(<<'END_SOLUTION'));
${PAR}РЕШЕНИЕ:${PAR}
Здесь должно быть отображено решение, но его нет.
END_SOLUTION
Context()->normalStrings;

COMMENT('MathObject version.');

ENDDOCUMENT();
```

Концовка содержит заготовку для отображения решения. Но само решение не подготовлено. Команда `COMMENT('MathObject version.');` также не оказывает влияния на отображение задачи в песочнице.

14. Задача с матрицами с одним текстовым окном ввода для ответа.

Начало задачи стандартное – загружаются макросы:

```
DOCUMENT();
loadMacros("PGstandard.pl","MathObjects.pl","PGcourse.pl",);
```

В процессе инициализации объектов задачи используется матричный контекст:

```
TEXT(beginproblem());
Context('Matrix');

$example1 = Matrix([[1,2,3],[4,5,6]]);
$example2 = $example1->column(1);
$example3 = $example1->row(1);
$example4 = Matrix([[1,2,3],[4,5,6]]);
```

Видим, что в матричном объекте имеются методы, позволяющие выделить из него отдельные столбцы и отдельные строки.

```
Context()->texStrings;
BEGIN_TEXT
Цель данного вопроса – показать синтаксис WeBWorK, используемый для записи
матриц, когда имеется всего одно окно ввода (что не очевидно) и когда име-
ется по одному окну ввода для каждого элемента матрицы (что очевидно).
Приведённые ниже примеры разъясняют это.
$BR
$BR
При записи матриц используются квадратные скобки для того, чтобы предста-
вить их элементы в виде списков. Матрица с одной строкой записывается как
```

```
список элементов, разделённых запятыми, и окружённый квадратными скобками.
Матрица с более чем одной строкой или одиночные столбцы изображаются как
списки из списков, каждый из которых отображает одну из строк матрицы.
Ваши ответы должны использовать квадратные скобки для каждой строки в мат-
рице и для каждого элемента в одиночном столбце. Кроме того, ещё одна пара
квадратных скобок нужна для окружения списка строк в матрице с более чем
одной строкой (например, в матрице размером \( 2 \times 3 \)) и в столбце
\( 2 \times 1 \), который имеет более одной строки. А в матрице \
( 1 \times 3 \) достаточно одной пары квадратных скобок. Ваш ответ может
содержать пробелы и переносы строк, как в следующем примере:
$PAR
$BCENTER
[ [1, 2, 3], $BR [4, 5, 6] ]
$ECENTER
$BR
$BR
Введите матрицу \( $example1 \) в виде строки \{ $example1->string \}
$BR
\{ ans_box(3,30) \}
$BR
$BR
Введите столбец \( $example2 \) в виде \{ $example2->string \}
$BR
\{ ans_box(3,30) \}
$BR
$BR
Введите строку \( $example3 \) в виде \{ $example3->string \}
$BR
\{ ans_box(3,30) \}
END_TEXT
Context()->normalStrings;
```

Выписанный выше текст задачи достаточно очевиден зам по себе. В нём важно лишь заметить, что у матричного объекта есть метод `->string`, позволяющий отобразить его в виде строки с квадратными скобками.

Проверка ответа выглядит стандартно, несмотря на использование матричных объектов и текстовый способ ввода ответа в одном окне на целую матрицу:

```
$showPartialCorrectAnswers = 1;

ANS( $example1->cmp() );
ANS( $example2->cmp() );
ANS( $example3->cmp() );
```

Текст задачи имеет продолжение:

```
Context()->texStrings;
BEGIN_TEXT
$BR $BR
Ввод матрицы с помощью матричной формы для ответа выглядит естественно.
$BR
Поместите по одному элементу матрицы \( $example4 \) в каждое из полей
ввода:
\{ $example4->ans_array \}
END_TEXT
Context()->normalStrings;
```

Но это не многошаговая задача. Все части текста задачи отображаются одновременно. Проверка второй части задачи и её концовка выглядят стандартно:

```
ANS( $example4->cmp() );
COMMENT('MathObject version.');
```

```
ENDDOCUMENT();
```

15. Матричная задача с собственным алгоритмом проверки.

В матричных задачах тоже может реализовываться алгоритм проверки, отличный от простой проверки совпадения. Здесь рассматривается пример такой задачи:

```
DOCUMENT();
loadMacros("PGstandard.pl","MathObjects.pl","parserMultiAnswer.pl",
"AnswerFormatHelp.pl","PGcourse.pl",);
$showPartialCorrectAnswers = 0;
```

Начало задачи, как видим, стандартное – загружаются макросы:

```
TEXT(beginproblem());
Context('Matrix');

$A = Matrix([[1,1],[0,1]]);
$B = Matrix([[1,0],[1,1]]);

$multians = MultiAnswer($A, $B)->with(
  singleResult => 1,
  checker => sub {
      my ( $correct, $student, $answerHash ) = @_;
      my @s = @{$student};
      $s0 = Matrix($s[0]);
      $s1 = Matrix($s[1]);
      return $s0 * $s1 != $s1 * $s0;
  }
);
```

При инициализации задачи используется матричный контекст и объект многошаговости. В объекте многошаговости содержится процедура проверки, Матрицы `$A` и `$B` передаются в объект `$multians` в качестве правильных ответов. Но они не являются единственным правильным ответом, поэтому в проверке они фактически не используются. Здесь из ответов, введённых студентом `@{$student}`, формируются два новых матричных объекта `$s0` и `$s1`. Ответ считается правильным, если матрицы `$s0` и `$s1` не перестановочны:

```
Context()->texStrings;
BEGIN_TEXT
Приведите пример двух матриц \( A \) и  \( B \) размером \( 2 \times 2 \),
таких что \( AB \ne BA \).
$BR
$BR
\( A = \)
\{ $multians->ans_array(5) \}
\ \ \ \
\( B = \)
\{ $multians->ans_array(5) \}
END_TEXT
Context()->normalStrings;
```

В приведённом выше тексте задачи видим, что объект `$multians` наследует от своих матричных аргументов `$A` и `$B` метод `ans_array(5)` и дважды применяет этот метод для того, чтобы получить от студента две матрицы `$s0` и `$s1`.

Процедура проверки выглядит стандартно:

```
install_problem_grader(~~&std_problem_grader);
```

```
ANS( $multians->cmp() );
COMMENT('MathObject version.');

ENDDOCUMENT();
```

Строка относится к тому, как оценивается задача в случае, когда не все ответы в бланках ответов верные. Подробности можно найти здесь.

16. Задача на матричные операции.

Подобно предыдущим матричным задачам, эта задача начинается стандартно:

```
DOCUMENT();
loadMacros("PGstandard.pl","MathObjects.pl","parserPopUp.pl",
           "AnswerFormatHelp.pl","PGcourse.pl",);
$showPartialCorrectAnswers = 0;
```

Задача начинается с создания двух объектов типа матрица и объекта типа `PopUp`:

```
TEXT(beginproblem());
$A = Matrix([[non_zero_random(-5,5,1),non_zero_random(-5,5,1)],
[non_zero_random(-5,5,1),non_zero_random(-5,5,1)],]);

$B = Matrix([[non_zero_random(-5,5,1),non_zero_random(-5,5,1),
non_zero_random(-5,5,1)],[non_zero_random(-5,5,1),
non_zero_random(-5,5,1),non_zero_random(-5,5,1)],]);

$popup = PopUp(['Choose','True','False'],'False');
```

Вывод текста задачи для студентов использует матричный когтекст:

```
Context()->texStrings;
BEGIN_TEXT
Пусть
\[ A = $A,\]
\[ B = $B.\]
Если это возможно, посчитайте следующие величины. Если же это невозможно,
введите ${BBOLD}DNE${EBOLD} в окнах ввода ответа.
$BR
$BR
\( AB = \)
\{ ans_box(3,30).$SPACE.AnswerFormatHelp('matrices') \}
$BR
$BR
\( BA = \)
\{ ans_box(3,30).$SPACE.AnswerFormatHelp('matrices') \}
$BR
$BR
\{ $popup->menu \} Верно (True) или  неверно (False), что для любых двух
матриц \( A \) и \( B \) оба произведения \( AB \) и \( BA \) всегда опре-
делены.
END_TEXT
Context()->normalStrings;
```

Особенностью задачи является то, что для ввода матричного ответа используется простое текстовое окно ввода. Проверка ответа также имеет особенности:

```
install_problem_grader(~~&std_problem_grader);
ANS( ($A * $B)->cmp() );
ANS( Compute('DNE')->cmp() );
```

```
ANS( $popup->cmp );
COMMENT('MathObject version.');

ENDDOCUMENT();
```

Первое из матричных произведений определено и для проверки используется метод `cmp()` матричного объекта, который конструируется тут же путем перемножения матриц `($A * $B)`. Второе произведение не определено. Поэтому здесь используется объект типа текстовая строка, который формируется из текста `'DNE'` при помощи функции `Compute`. Строка `install_problem_grader(~~&std_problem_grader);` относится к оценке результата, а не к проверке ответа непосредственно.

17. Добавление слайд-шоу и роликов с YouTube в задачу.

Задачи в WebWork можно разнообразить, добавляя в них презентации из Google и видеоролики из YouTube. Рассмотрим код одной из таких задач:

```
DOCUMENT();
loadMacros("PGstandard.pl","MathObjects.pl","AnswerFormatHelp.pl",
           "PGcourse.pl",);
```

Начало, как видим стандартное и никаких необычных макросов не загружается. Далее используется функция создания объекта `MODES`. Ей передаются два параметра `HTML` и `TeX`. Помимо прочего, этот приём подсказывает, как вставлять тэги `HTML` в задачу:

```
TEXT(beginproblem());
Context("Numeric");

$slideshow = MODES(
HTML=>
"<iframe
src='https://docs.google.com/presentation/d/1pk0FxsamBuZsVh1WGGmHGEb5AlfC6
8KUlz7zRRIYAUg/embed#slide=id.i0'
frameborder='0' width='555' height='451'></iframe>",
TeX =>
"An embedded Google slide show.");

$video = MODES(
HTML=>
'<iframe width="420" height="315"
src="https://www.youtube.com/embed/PnqUzaahAsQ"
frameborder="0" allowfullscreen></iframe>',
TeX =>
"An embedded YouTube video.");
```

Параметры `HTML` и `TeX` являются строками. Их синтаксис определяется не языком PG или Perl, а языком форматирования интернет-страниц `HTML`. Для отобоажения слайд-шоу и видео применяется тэг `<iframe>` с параметрами. Далее слкдует код для отображения построенных объектов `$slideshow` и `$video`.

```
Context()->texStrings;
BEGIN_TEXT
${BCENTER}
$slideshow
$BR
Слайды с Google встроены при помощи тэга iframe.
$BR
$BR
$video
```

```
$BR
Видео с YouTube, мультфильм “Буревестник”, встроен при помощи тэга iframe.
$BR
Если видео не сработало – кликните по ссылке
\{ htmlLink("https://www.youtube.com/embed/PnqUzaahAsQ",
"и перейдите непосредственно на YouTube.") \}
${ECENTER}
END_TEXT
Context()->normalStrings;
```

Как видим, отображение слайдов презентации и видео происходит путем упоминания имён соответствующих объектов в тексте задачи. Важно отметить, пример встраивания ссылки `htmlLink()`.

```
$showPartialCorrectAnswers = 1;

Context()->texStrings;
SOLUTION(EV3(<<'END_SOLUTION'));
${PAR}SOLUTION:${PAR}
Здесь должно быть решение.
END_SOLUTION
Context()->normalStrings;

COMMENT('MathObject version.');

ENDDOCUMENT();
```

Концовка стандартная. Она содержит заготовку для решения, хотя в данном примере решение не нужно, ибо текст задачи лишь демонстрирует возможности WebWork.

18. Использование аплетов Flash анимации.

Это задача на построение графика возрастающей функции при помощи аплета Flash анимации. Задача имеет подробный заголовок. В заголовке имеется указание на то, что составитель задачи `Barbara Margolius` получала финансовую поддержку Национального Научного Фонда (NSF) по гранту с номером `0941388`:

```
## DESCRIPTION
## understanding derivatives graphically
## ENDDESCRIPTION
## KEYWORDS('derivatives', 'graph','Flash applets','NSF-0941388')
## DBsubject('Calculus')
## DBchapter('Limits and Derivatives')
## DBsection('Derivatives')
## Date('6/12/2012')
## Author(Barbara Margolius')
## Institution('Cleveland State University')
## TitleText1('')
## EditionText1('2012')
## AuthorText1('')
## Section1('')
## Problem1('')
```

Сама задача начинается с загрузки соответствующих макросов:

```
DOCUMENT();
loadMacros("PGanswermacros.pl", "PGstandard.pl", "AppletObjects.pl",
```

```
            "MathObjects.pl",);
```

Далее следует блок инициализации переменных

```
TEXT(beginproblem());
$showPartialCorrectAnswers = 1;
Context("Numeric");

$ans =Compute("1");

$pos = 2;
$inc = 1;
$cup = 2;

$boardMessage = "Sketch a function with a positive derivative.";
$showMM = 'false'; # don't display x and y ranges
```

Следующий шаг – поиск аплета по имени и создание объекта типа аплет:

```
$appletName = "graphSketch";
$applet =  FlashApplet(
   codebase              => findAppletCodebase("$appletName.swf"),
   appletName            => $appletName,
   appletId              => $appletName,
   setStateAlias         => 'setXML',
   getStateAlias         => 'getXML',
   setConfigAlias        => 'setConfig',
   getConfigAlias        => 'getConfig',
   maxInitializationAttempts => 5,
   answerBoxAlias        => 'answerBox',
   height                => '550',
   width                 => '550',
   bgcolor               => '#ededed',
   debugMode             =>  0,
   submitActionScript  =>
  qq{getQE("answerBox").value=getApplet("$appletName").getAnswer()},);
```

Далее следует блок инициализации данных аплета. Если `$pos=0`, то аплет требует отрицательных значений функции, если `$pos=1`, то аплет требует положительных значений функции. В нашем случае выше было выбрано `$pos=2`. Это значит, что аплет не реагирует на знак функции.

Если `$inc=0`, то аплет требует убывающей функции, если `$inc=1`, то аплет требует возрастающей функции. Если `$inc=2`, то аплет не реагирует на характер роста функции. В нашем случае выше было выбрано `$inc=0`.

Если `$cup=0`, то аплет требует вогнутой вниз функции, если `$cup=1`, то аплет требует вогнутой вввкпх функции. Если `$cup=2`, то аплет не реагирует на характер вогнутости функции. В нашем случае выше было выбрано `$inc=1`.

От студента также может требоваться чтобы графие функции проходил через одну или две заданные точки. Запись `<pts><pt xval='1' yval='2' showIt='true'/></pts>` будет означать, что отобразится точка с координатами `(1,2)`, и надо будет провести график через неё.

Параметр `$boardMessage` содержит текстовое сообщение, отображаемое аплетом. Параметр `$showMM` отображает диапазон изменения `x` и `y` в аплете.

```
$applet->configuration(qq{<xml><pos>$pos</pos><inc>$inc</inc>
<cup>$cup</cup><boardMessage>$boardMessage</boardMessage>
<showMM>$showMM</showMM><bland>true</bland></xml>});
```

```
$applet->initialState(qq{<xml><pos>$pos</pos><inc>$inc</inc>
<cup>$cup</cup><boardMessage>$boardMessage</boardMessage>
<showMM>$showMM</showMM><bland>true</bland></xml>});

TEXT(MODES(TeX=>'object code', HTML=>$applet->insertAll(
  debug=>0,includeAnswerBox=>1,
  # reinitialize_button=>$permissionLevel>=10,
)));
```

Возможность отображения кнопки переинициализации отключена. Эта орция закомментарена при помощи значка решётки.

```
TEXT(MODES(TeX=>"", HTML=><<'END_TEXT'));
<script>
if (navigator.appVersion.indexOf("MSIE") > 0) {
document.write("<div width='3in' align='center'
style='background:yellow'>По видимому Вы используете
Internet Explorer.<br/> Рекомендуется перейти на другой браузер при про-
смотре этой страницы.</div>");}
</script>
END_TEXT
```

Приведённый выше код вводит в отображаемую на экране студента страницу скрипт на языке, который проверяет, не используется ли браузер Internet Explorer, и предупреждает, если он используется.

```
BEGIN_TEXT
$BR $boardMessage

$BR Нажмите 'score', чтобы проверить вашу работу. Если она успешна
(SUCCESS), нажмите кнопку 'submit answers'.

$BR Если ваш график близок к правильному, то есть не очень большая его
часть выделена красным, кнопка 'SMOOTH' может поправить его.

END_TEXT
Context()->normalStrings;
```

Зачётная проверка ответа идёт через функцию `NAMED_ANS()`. В ранее рассмотренных примерах использовалась функция `ANS()`. Дело в том, что ответ здесь формируется через аплет, а не через окно ввода ответа.

```
NAMED_ANS('answerBox'=>$ans->cmp());
ENDDOCUMENT();
```

Концовка задачи по команде `ENDDOCUMENT();` стандартна. Отметим, что программирования самого аплета мы не касались. Это отдельное искусство, которое описано здесь.

19. Всплывающие подсказки Knowl.

Иногда бывает нужно дать студенту дополнительную информацию и при этом не загромождать текст задачи. Для этой цели служат подсказки Knowl. Рассмотрим пример

```
DOCUMENT();
loadMacros("PGstandard.pl","MathObjects.pl",);
```

Никаких специальных макросов для подсказок Knowl не требуется. Поэтому начало стандартное. Переходим к тексту задачи. Первый тип подсказок содержит просто текст.

```
TEXT(beginproblem());
```

```
Context()->texStrings;
BEGIN_TEXT
$BR
Это подсказка, использующая технологию knowl:
\{ knowlLink("кликни меня",
value=>'Это внутренность подсказки. '.
'Если кликнуть подсказку ещё раз, она закроется') \}
```

Второй тип подсказок содержит Интернет ссылку.

```
$BR
Это другая подсказка, использующая технологию knowl, она загружает внешний
файл:
\{ knowlLink(“кликни меня”,
url=>’http://freetextbooks.narod.ru/rsharipov/r16.htm’) \}
```

Третий тип подсказок содержит текст со встроенной в него математикой

```
$BR
Это подсказка, содержащая математический текст:
\{ knowlLink(“кликни меня”,
value=>escapeSolutionHTML(EV3P(“Внимание, это функция синус”.
“\\(\sin(x)\\)”)), base64=>1) \}
END_TEXT
Context()->normalStrings;

ENDDOCUMENT();
```

20. Всплывающие подсказки HINT и подсказки к ответу AnswertHints.

Подсказки HINT отличаются от подсказок Knowl тем, что они становятся доступными после нескольких неудачных попыток дать правильный ответ. Разберём пример.

```
DOCUMENT();
loadMacros("PGstandard.pl","MathObjects.pl","answerHints.pl",);
```

Начало стандартное, но загружен макрос `answerHints.pl`. Текст этого макроса можно посмотреть здесь. Но это нужно лишь для технических специалистов.

Далее идёт инициализация переменных. Она стандартная. Но заметим, в контексте уточнён тип переменных `Real`.

```
TEXT(beginproblem());

Context("Numeric");
Context()->variables->are(t=>"Real",u=>"Real");

$f = Formula("2 t");
$answer = Formula("6 u");
```

После инициализации переменных идёт отображение текста задачи.

```
Context()->texStrings;
BEGIN_TEXT
If \( f(t) = $f \), then \( f(3u) \) = \{ ans_rule(10) \}
$BR
$BR
\{
knowlLink("Кликните, чтобы увидеть подсказку",
value=>'Подставьте 3u везде, где вы видите t в формуле для f.')
```

```
\}
END_TEXT
```

Здесь можно видеть подсказку `Knowl`, с которой мы уже познакомились выше. Подсказка `HINT` формируется ниже.

```
$showHint = -1;
BEGIN_TEXT
$PAR
Если у Вас не получиться дать правильный ответ с $showHint попыток, Вы по-
лучите подсказку.
END_TEXT
HINT(EV3(<<'END_HINT'));
Повторяю, подставьте \( 3u \) везде, где вы видите \( t \) в формуле для \
( f(t) = $f \).
END_HINT
Context()->normalStrings;
```

Важное значение имеет переменная `$showHint`. Её значение указывает, после скольких попыток будет показана подсказка. В данном примере это значение равно -1. Дело в том, что пример предназначен для отработки в песочнице on-line PG labs. В этом случае подсчёт попыток не работает в силу настроек этой песочницы.

```
$showPartialCorrectAnswers = 1;

ANS($answer->cmp()->withPostFilter(AnswerHints(
  Formula("6 t") => "А правильную ли переменную Вы используете?",
  Formula("6 u") => "Это правильно. Отличная работа!",
)));

ENDDOCUMENT();
```

Подсказка `AnswertHints` программируется в области проверки ответов. Это видно в приведённом выше фрагменте программы.

Заметим, что в рассмотренном выше примере подсказка `HINT` отображает заголовок Hint на английском языке. Это можно переделать, если модифицировать подпрограмму `HINT_HEADING`, как показано ниже.

```
sub HINT_HEADING { MODES(
TeX => "\\par {\\bf Подсказка:}",
Latex2HTML => "\\par {\\bf Подсказка:}",
HTML => "<P><B>Подсказка:</B>"); };
```

21. Подсказки PANIC, снижающие оценку.

В некоторых случаях уровень подсказки может быть очень высок и составитель задачи может предусмотреть определённый штраф за использование такой подсказки. Разберём пример, полученный из предыдущего примера добавлением подсказок `PANIC`.

```
DOCUMENT();
loadMacros("PGstandard.pl","MathObjects.pl","answerHints.pl",
           "problemPanic.pl");

TEXT(beginproblem());

Context("Numeric");
Context()->variables->are(t=>"Real",u=>"Real");
```

```
$f = Formula("2 t");
$answer = Formula("6 u");
```

Начало, как видим, такое же, как в предыдущем примере. Но добавлена загрузка одного макроса `problemPanic.pl`.

```
Context()->texStrings;
BEGIN_TEXT
If \( f(t) = $f \), then \( f(3u) \) = \{ ans_rule(10) \}
$BR
$BR
\{
knowlLink("Кликните, чтобы увидеть подсказку Knowl",
value=>'Подставьте 3u везде, где вы видите t в формуле
для f.')
\}
$BR
END_TEXT
```

Эта часть кода выводит на экран студента текст задача и подсказку `Knowl`. Дальнейший вывод на экран студента зависит от значения переменной `$panicked`.

```
if ($panicked == 0) {
BEGIN_TEXT
Если Вам нужна более подробная подсказка, нажмите
кнопку
END_TEXT
}
```

Значение переменной `$panicked` первоначально равно нулю. Оно увеличивается на единицу после нажатия на кнопку Panic::Button(). Сама эта кнопка отображается на экране следующим кодом.

```
BEGIN_TEXT
\{Panic::Button(label => "Запросить подсказку", penalty => 0.25)\}
END_TEXT
```

Отображение на экране кнопки Panic::Button() не меняет величину переменной `$panicked`. Поэтому следующий текст тоже выводится на экран студента в первоначальном тексте задачи.

```
if ($panicked == 0) {
BEGIN_TEXT
, но знайте, что при этом вы потеряете 25% баллов за эту задачу.
$BR
END_TEXT
}
```

Текст подсказки `PANIC` отображается только в том случае, если кнопка Panic::Button() нажата и значение переменной `$panicked` стало равным 1.

```
if ($panicked > 0) {
BEGIN_TEXT
Подсказка за 25%:$BR Если в \(2 t\) вместо \(t\) подскавить \(3 u\), то
сколько это будет, а?
$BR$BR
END_TEXT
}
```

Этот текст может содержать вставки математических формул в формате TeX. Нажатие кнопки Panic::Button() обрабатывается через обращение к серверу. При этом отображение текста задачи студенту изменяется в соответствии с изменившимся значением переменной `$panicked`.

```
if ($panicked == 1) {
BEGIN_TEXT
Если Вам нужна ещё более подробная подсказка, нажмите кнопку
END_TEXT
}
```

Приведённый выше текст отображается внутри первой подсказки и готовит отображение второй кнопки Panic::Button().

```
if ($panicked > 0) {
BEGIN_TEXT
\{Panic::Button(label => "Запросить вторую подсказку", penalty => 0.50)\}
END_TEXT
}
```

Далее следует комментарий к отображённой кнопке.

```
if ($panicked == 1) {
BEGIN_TEXT
, но знайте, что при этом вы потеряете в сумме 25% + 25% = 50%
баллов за эту задачу.
$BR
END_TEXT
}
```

Текст второй подсказки `PANIC` отображается только в том случае, если нажата вторая кнопка Panic::Button() и значение переменной `$panicked` стало равным 2. При желании в текст этой подсказки можно добавить отображение третьей кнопки Panic::Button(). Число вложенности подсказок с кнопками ничем не ограничивается.

```
if ($panicked > 1) {
BEGIN_TEXT
Подсказка за 50%:$BR Если в \(2 t\) вместо \(t\) подскавить \(3 u\), то
это будет
\(6 t\) или \(6 u\)? Догадайтесь, какой из ответов правильный.
END_TEXT
}
Context()->normalStrings;
```

Концовка задачи похожа на концовку предыдущей задачи.

```
$showPartialCorrectAnswers = 1;
Panic::GradeWithPenalty();

ANS($answer->cmp()->withPostFilter(AnswerHints(
Formula("6 t") => "А правильную ли переменную Вы используете?",
Formula("6 u") => "Это правильно. Отличная работа!",
)));

ENDDOCUMENT();
```

Важным отличием является команда `Panic::GradeWithPenalty()`. Она включает механизм понижения оценки за задачу при использовании подсказок типа `PANIC`.

22. Использование системы символьных вычислений SAGE при помощи интерфейса AskSage.

WebWork обладает определённым функционалом для символьных вычислений. Например, выше в разделе 7 мы видели способность объектов типа `Formula` к дифференцированию при помощи метода `=D()`. Но все равно, этот функционал значительно слабее возможностей специализированных пакетов типа `Waterloo Maple` и `Wolfram Matematika`. Указанные пакеты являются коммерческими. Но имеется бесплатный пакет символьных вычислений `SAGE` (см. http://www.sagemath.org/ и http://www.sagemath.org/ru/). WebWork имеет доступ к on-line версии пакета `SAGE` через процедуру `AskSage()`. Для использования возможностей важно уметь передавать данные из WebWork в `SAGE` и обратно. Рассмотрим пример реализующий эти действия.

```
DOCUMENT();
loadMacros("PGstandard.pl","MathObjects.pl",);
```

Начало стандартное. Оно даже не содержит каких-то специальных макросов. Функция `AskSage()` доступна в одном из стандартных макросов.

```
TEXT(beginproblem());
$WebWorkVariable=7;
```

В этом фрагменте кода определяется переменная `$WebWorkVariable`, которой присваивается числовое значение 7.

```
$SageCode=<<END_SAGE;
SageVariable = $WebWorkVariable
WEBWORK['variable'] = SageVariable
END_SAGE
```

В этом фрагменте кода определяется строковая переменная `$SageCode`. Она содержит две строки текста, заключенные между объявлением метки `<<END_SAGE;` и самой меткой `END_SAGE`. Эта строковая переменная представляет собой код, написанный на языке `SAGE`. При отображении в этой строке переменная $WebWorkVariable заменяется её числовым значением 7. Далее следует код

```
BEGIN_TEXT
Это код, который будет исполняться в SAGE:
$BR
$SageCode
$BR
$BR
END_TEXT
```

Этот легко понимаемый код отображает строковую переменную `$SageCode` и позволяет в явной форме увидеть код, который будет исполняться в `SAGE`.

```
SageVariable = 7 WEBWORK['variable'] = SageVariable
```

Даже не зная языка `SAGE`, легко понять этот код. Переменной **`SageVariable`** присваивается значение 7, переданное из переменной `$WebWorkVariable` при её отображении в строке `$SageCode`. Затем в специальную hash-таблицу с именем **`WEBWORK`** подставляется значение 7, совпадающее со значением переменной **`SageVariable`**. Hash-тадлицы – это массивы, индексированные не числами, а ключами. В данном случае ключом служит текстовая строка **`'variable'`**.

Следующий фрагмент кода самый важный.

```
$SageReturn = AskSage($SageCode,{seed=>$problemSeed,
                                 accepted_tos=>'true',
                                 debug=>0});
```

Здесь вызывается процедура `AskSage`. Её первый аргумент – это строка с кодом на языке `SAGE`. Второй аргумент – это hash-таблица с опциями. Функция создаёт объект `$SageReturn`. Это уже объект в WebWork. Он имеет поле с именем `webwork`, который указывает на другой объект, у которого имеется поле `variable`. В этом поле и передаётся значение 7, выработанное в коде `SAGE`. Фрагмент программы

```
if ( sageReturnedFail($SageReturn)  ){
 $SageReturn->{webwork}->{variable}= "u";
}
```

отрабатывает особый случай, когда в коде `SAGE` произошла ошибка. Он заменяет возвращаемое значение 7 на строку `"u"`, что может использоваться далее как сигнал об ошибке. Если же ошибки не было, срабатывает оставшийся фрагмент программы

```
$webwork_objects = $SageReturn->{webwork};
$R = Real($webwork_objects->{variable});
BEGIN_TEXT
Значение, полученное из SAGE, равно $R
END_TEXT
ENDDOCUMENT();
```

Он превращает число 7, полученное из `SAGE` в объект WebWork типа Real и печатает в тексте задачи.

23. Пример задачи, в которой ответом является уравнение.

Начало этой задачи стандартное. Загружаются необходимые макросы. Среди прочих загружается макрос `parserAssignment.pl`. Он нужен для того, чтобы использовать знак равенства в формулах, превращая их в уравнения.

```
DOCUMENT();

loadMacros(
"PGstandard.pl",
"MathObjects.pl",
"AnswerFormatHelp.pl",
"answerHints.pl",
"parserAssignment.pl",
);
```

Далее выбирается числовой контекст, в котором добавляется одна переменная `y`:

```
TEXT(beginproblem());
Context("Numeric")->variables->add(y=>"Real");
parser::Assignment->Allow;
```

Добавление переменной в контекст мы уже встречали выше (см. раздел 8). Новой является команда `parser::Assignment->Allow;`. Она позволяет использовать знак равенства в формулах.

```
$a = random(2,5,1);
```

```
$aa = $a**2;
$a2 = 2 * $a;

$f = Compute("sqrt(x)");
$answer = Compute("y = $a + (1/$a2) * (x-$aa)");
```

В этом блоке команд инициализируются переменные. Среди них переменная `$answer`, с которой будет производиться сравнение ответа, который введёт студент. Важно отметить, что ей присваивается формула, содержащая знак равенства, который превращает её в уравнение.

```
Context()->texStrings;
BEGIN_TEXT
Импользуя тейлоровское разложение, найдите линейное приближение для функции \( f(x) = $f \) в окрестности точки \( x = $aa \).  Ваш ответ должен быть записан как равенство в переменных \( x \) и \( y \).
$BR
$BR
\{ ans_rule(20) \}
\{ AnswerFormatHelp("equations") \}
END_TEXT
Context()->normalStrings;
```

Это текст задачи, отображаемый в браузере у студента. К окну для ввода ответа прилагается подсказка, которая отображается в виде гиперссылки. Эта гиперссылка отображает подсказку по теме "equations", что значит уравнения. Выше мы уже встречали такую подсказку по теме "matrices". Ниже приведён список некоторых возможных тем в таких подсказках.

```
AnswerFormatHelp("syntax")
AnswerFormatHelp("matrices")
AnswerFormatHelp("numbers")
AnswerFormatHelp("equations")
AnswerFormatHelp("inequalities")
```

Вновь возвращаемся к коду программы.

```
$showPartialCorrectAnswers = 1;

ANS( $answer->cmp()
->withPostFilter(AnswerHints(
  [Formula("1/$a2"),Formula("y=1/$a2")] =>
  ["Ваш ответ должен быть уравнением негоризонтальной прямой линии.",
  replaceMessage=>1],
)));
ENDDOCUMENT();
```

Проверка ответа также снабжена подсказками. В разделе 20 выше мы уже видели такие подсказки. В данном случае имеется дополнительная опция `replaceMessage=>1`. Но при прогонке в песочнице on-line PG labs влияние этого параметра выявить не удаётся.

24. Задача с развёрнутым текстовым ответом, которая оценивается вручную.

Начало задачи стандартное. Но помимо стандартных макросов, здесь загружаются ма-

кросы parserPopUp.pl и PGessaymacros.pl.

```
DOCUMENT();
loadMacros("PGstandard.pl","MathObjects.pl","parserPopUp.pl",
                    "PGessaymacros.pl","PGcourse.pl",);
```

Следующий затем код демонстрирует загрузку особого средства для оценки ответов:

```
TEXT(beginproblem());
$showPartialCorrectAnswers = 0;
install_problem_grader(~~&std_problem_grader);
```

Следующий затем код демонстрирует загрузку особого средства для оценки ответов: Дальнейший код работает в числовом контексте

```
Context("Numeric");
$popup = PopUp(["Choose","True","False" ],"False");
```

Дальнейший код работает в числовом контексте. Инструмент PopUp() мы уже видели в разделе 12. Он даёт выбор из нескольких вариантов в окне выбора.

```
$a = random(2,5,1);
$f1 = Compute("ln(x (x-$a))");
$f2 = Compute("ln(x) + ln(x-$a)");
```

Этот код генерирует одно случайное число и два объекта типа формула, в котором это число используется.

```
Context()->texStrings;
BEGIN_TEXT
Ответьте на вопрос верно или неверно сформулированное ниже утверждение и
обоснуйте Ваш ответ. Ваш развёрнутый ответ будет оцениваться преподава-
телем вручную.
$BR
$BR
\{ $popup->menu() \}
Верно или неверно, что для всех вещественных чисел \( x \), выполнено ра-
венствиво \( $f1 = $f2 \).
$BR
$BR
Объясните ваш выбор словами в ячейке для развёрнутого ответа.
$BR
\{ essay_box(8,60) \}
END_TEXT
Context()->normalStrings;
```

Этот код отображает текст задачи. Здесь мы видим отображение инструмента выбора PopUp() и отображение ячейки для развёрнутого ответа essay_box(8,60). Заметим, что такая ячейка отличается от ячейки для текстового ответа ans_box(3,30), которую мы видели в разделе 14 выше.

```
ANS($popup->cmp());
ANS(essay_cmp());
ENDDOCUMENT();
```

Проверка ответа состоит из двух частей. Первая часть стандартна. Вторая часть использует функцию essay_cmp(), которая не связана ни с одним из объектов. Это наследие старых времен, когда все ответы проверялись необъектными процедурами. Еще одна особен-

ность – это наличие на экране кнопки с текстом “Show Problem Sourse”. Эта кнопка отображается нестандартным средством проверки, которое загрузилось выше при помощи команды `install_problem_grader(~~&std_problem_grader);`.

25. Задача, в которой строится график функции с закрашенной областью под ним.

Начало этой задачи тоже стандартное. Использование макроса мы уже видели в разделе 9. Здесь мы рассмотрим ещё один пример использования этого макроса.

```
DOCUMENT();
loadMacros("PGstandard.pl","MathObjects.pl","AnswerFormatHelp.pl",
           "PGgraphmacros.pl","unionTables.pl",);
```

Далее следует код, активирующий опцию обновления загруженных изображений:

```
TEXT(beginproblem());
$refreshCachedImages = 1;
```

В инициализационной части задачи выбирается одно случайное число и одна функция. Ответом является число, которое получается в результате вычислений:

```
Context("Numeric");
$a = random(0,3,1);
$f = Formula("sqrt(x)+$a");
$answer = Compute("(2/3) * (4^(3/2) - 1) + 3*$a");
```

Создание основы для графика, задание диапазонов, осей, координатной сетки:

```
$gr = init_graph(-5,-5,5,5,grid=>[10,10],axes=>[0,0],pixels=>[300,300]);
```

Создание меток на графике

```
$gr->lb('reset');
foreach my $j (1..4) {
  $gr->lb( new Label(-4.7,  $j, $j,'black','center','middle'));
  $gr->lb( new Label(-4.7, -$j,-$j,'black','center','middle'));
  $gr->lb( new Label(  $j,-4.7, $j,'black','center','middle'));
  $gr->lb( new Label( -$j,-4.7,-$j,'black','center','middle'));
}
$gr->lb( new Label(4.7,0.2,'x','black','center','middle'));
$gr->lb( new Label(0.2,4.7,'y','black','center','middle'));
```

Заметим, что сами метки являются объектами. Далее определяется палитра цветов. Цвета тоже являются объектами:

```
$gr->new_color("lightblue", 214,230,244);
$gr->new_color("darkblue",  100,100,255);
$gr->new_color("lightgreen",156,215,151);
$gr->new_color("darkgreen",   0, 86, 34);
$gr->new_color("lightred",  245,234,229);
$gr->new_color("darkred",   159, 64, 16);
```

Названия цветов присваиваются переменным

```
$light = "lightblue";
$dark = "darkblue";
```

Названия цветов присваиваются переменным . Далее к графику добавляется функция:

```
add_functions($gr,"$f  for x in <0,5> using color:$dark and weight:2");
```

Отметим, что в разделе 9 использовалась функция `plot_functions`. Её синтаксис идентичен синтаксису функции `add_functions` здесь. В дополнение к графику функции задаются три отрезка прямых линий, образующих ломаную линию.

```
$gr->moveTo(1,$a+1);
$gr->lineTo(1,0,$dark,2);
$gr->lineTo(4,0,$dark,2);
$gr->lineTo(4,$a+2,$dark,2);
```

Функция закраски области под графиком функции заложена в объекте типа график:

```
$gr->fillRegion([1.1,0.1,$light]);
```

В отображаемом тексте задачи видим текстовую строку, в которую вписана ячейка для ввода числового значения и подсказка к этой ячейке.

```
Context()->texStrings;
BEGIN_TEXT
\{
ColumnTable(
"Используя график, найдите площадь закрашенной области под графиком функции \( f(x) = $f \).
$BR
$BR
Площадь = " .
ans_rule(20).$SPACE.
AnswerFormatHelp("numbers"),
image( insertGraph($gr),height=>300,width=>300,tex_size=>800 ).
$BR.$BCENTER.
$BR.
"График функции \( y = f(x) \)".
$ECENTER
,
indent => 0, separation => 30, valign => "TOP"
)
\}
END_TEXT
Context()->normalStrings;
```

Под ней текстовая строка, в которую встроен сам график в виде картинки и подпись под графиком. Эти две текстовые строки встроены в объект типа таблица.

```
$showPartialCorrectAnswers = 1;
ANS( $answer->cmp() );

ENDDOCUMENT();
```

Проверка задачи состоит в проверке совпадения одного числового значения.

26. Задача, в которой строится таблица значений функции.

Начало этой задачи стандартное. Никаких незнакомых макросов в этой задаче нет.

```
DOCUMENT();
loadMacros("PGstandard.pl","MathObjects.pl","AnswerFormatHelp.pl",);
```

Инициализационная часть задачи немного необычна, хотя достаточно понятна.

```
TEXT(beginproblem());

Context("Numeric");
$f = Formula("3^(-x)");

@answer = ();
foreach my $i (0..2) {
  $answer[$i] = $f->eval(x=>$i);
}
```

В ней создаётся пустой массив из ответов. Затем он в цикле заполняется значениями, полученными подстановкой переменной цикла в аргумент функции `$f`.

Текст, отображаемый на экране студента тоже довольно стандартный, если не считать таблицы, которая создается специальными командами `begintable(5)` и `endtable()`:

```
Context()->texStrings;
BEGIN_TEXT
Дана функция \( f(x) = $f \). Заполните числами таблицу значений  этой
функции.
\{ AnswerFormatHelp("numbers") \}
$PAR
$BCENTER
\{
begintable(5) .
row( "\(x = \)", "0", "1", "2" ) .
row( "\(f(x) = \)", ans_rule(5), ans_rule(5), ans_rule(5) ) .
endtable();
\}
$ECENTER
END_TEXT
Context()->normalStrings;
```

Эти команды записаны в строковом контексте TeX. Это означает, что они генерируют текст, содержащий тэги TeXa.

```
$showPartialCorrectAnswers = 1;
foreach my $i (0..2) {
  ANS( $answer[$i]->cmp() );
}

ENDDOCUMENT();
```

Концовка задачи примечательна тем, что проверка ответа в ней тоже заключена внутрь оператора цикла.

27. Задача с ответами, которые могут вводиться в произвольном порядке.

Начало этой задачи стандартное. Но в нём имеется загрузка нового для нас макроса `unorderedAnswer.pl`.

```
DOCUMENT();
loadMacros("PGstandard.pl","MathObjects.pl","AnswerFormatHelp.pl",
                    "unorderedAnswer.pl",);.
```

В инициализационной части используется числовой контекст и к нему добавляются две вещественные переменные `y` и `z`.

```
TEXT(beginproblem());

Context("Numeric")->variables->add(y=>"Real",z=>"Real");
$a = random(2,9,1);
$answer1 = Compute("x^$a");
$answer2 = Compute("y^$a");
$answer3 = Compute("z^$a");
```

Далее идёт отображаемый текст задачи.

```
Context()->texStrings;
BEGIN_TEXT
Запишите следующее выражение, раскрыв скобки. Запишите Ваш ответ в макси-
мально упрощённом виде, предполагая, что все переменные положительны.
$BR
$BR
\( (xyz)^{$a} = \)
\{ ans_rule(5) \}
\( \cdot \)
\{ ans_rule(5) \}
\( \cdot \)
\{ ans_rule(5) \}
\{ AnswerFormatHelp("formulas") \}
END_TEXT
Context()->normalStrings;
```

Ничего необычного в этой части задачи не наблюдается.

```
$showPartialCorrectAnswers = 1;

UNORDERED_ANS(
$answer1->cmp(),
$answer2->cmp(),
$answer3->cmp(),
);
ENDDOCUMENT();
```

А вот в концовке видим проверку задачи при помощи функции `UNORDERED_ANS()`, которая позволяет вводить ответы в произвольном порядке. Нечто подобное было реализовано в разделе 11, но гораздо более сложным способом.

28. Задача, иллюстрирующая понятие обратной функции при помощи Flash-аплета.

Начало этой задачи тоже стандартное. Такие начала мы уже видели.

```
DOCUMENT();
loadMacros( "PGstandard.pl",  "AppletObjects.pl", "MathObjects.pl");
```

После неё идёт инициализационная часть.

```
@funcArray = (
  "1/4*x^3-1",
  "x+cos(x)",
  "1/4*x^3-2*x",
  "4*atan(x)",
  "x*atan(x)",
  "1/2*x^(2)",
  "x-(x/2)^3"
```

```
);

@dispArray = (
  "\frac{1}{4} x^3 - 1",
  "x+\cos(x)",
  "\frac{1}{4} x^3 - 2 x",
  "4 \tan^{-1}(x)",
  "x \tan^{-1}(x)",
  "\frac{1}{2}x^2",
  "x-(\frac{1}{2}x)^3"
);

$rand = random(0,6,1);
$func = @funcArray[$rand];

$f = Formula($func);
$yval = random(1,3,1);
$ans1 = $f->substitute(x=>$yval);
$fder = $f->D();
$ans2 = 1/($fder->substitute(x=>$yval));
```

В инициализационной части создаётся массив из шести функций, из которого делается случайный выбор одной функции. Для этой функции выбирается случайное значение аргумента, для которого вычисляются значение функции и значение её производной.

```
    $appletName = "InverseGraph";
  $applet =  FlashApplet(
     codebase
       => findAppletCodebase("$appletName.swf"),
     appletName            => $appletName,
     setStateAlias         => 'setXML',
     getStateAlias         => 'getXML',
     setConfigAlias        => 'setConfig',
     height                => '400',
     width                 => '350',
     bgcolor               => '#e8e8e8',
     debugMode             => 0,
     submitActionScript  =>",
);
```

Эти часть кода находит аплет по имени `InverseGraph` и настраивает его параметры.

```
$applet->configuration(qq{
  <XML><Vars func = '$func'/></XML>});
$applet->initialState(qq{
  <XML><Vars func = '$func'/></XML>});
```

Эта часть кода тоже относится к конфигурации аплета. А дальше уже отображаемый текст задачи.

```
Context()->texStrings;
TEXT(beginproblem());
BEGIN_TEXT
$PAR

\{ $applet->insertAll(debug=>0,
includeAnswerBox=>0,
reinitialize_button=>0,) \}
$PAR
```

```
Пусть \(g(x)\) – обратная функция для функции
\(f(x)=$dispArray[$rand]\). Вычислите \(g($yval)\), не вычисляя формулы
для for g(x), и затем посчитайте  \(g'($yval)\).

$PAR \(g($yval)\) =  \{ans_rule()\}
$PAR \(g'($yval)\) =  \{ans_rule()\}

END_TEXT
```

Проверочная часть задачи абсолютно стандартна. Но она написана в старом стиле без использования объектного программирования.

```
ANS(num_cmp($ans1));
ANS(num_cmp($ans2));
```

Вся сложность картинки и её динамика заключена в аплете. Исходный код аплета в задаче не доступен.