Topic:

Index	1
Title	Math Question Generation
Supervisor(s)	Dr. Zhiyuan Li
No. of Students	1 – 3
Required knowledge	MATH2003 Discrete Structure COMP3173 Compiler Construction
Description	This project intends to design and implement a generator which takes students' ID as input and randomly generate questions in some specific domain of mathematics. The generator is expected to be applicable in real assignments or quizzes for all common undergraduate math courses. Thus, the generator regards math as a language. Each generated question is a string in the language. And those strings will pass lexical analysis, syntax analysis, and semantic analysis.

Task: WeBWorK : TEST

目前 uic 的 webwork 使用的题库是 pre-defined 的，现在希望能够生成这种格式的题目代码，让学生做到点击做题后现场生成题目，而不是使用题库中预先定义的题目。

1. What is WebWork?

WeBWorK is an **online homework system** (open source).
WeBWorK -- Open Source Online Homework System · GitHub

2. Core Features:

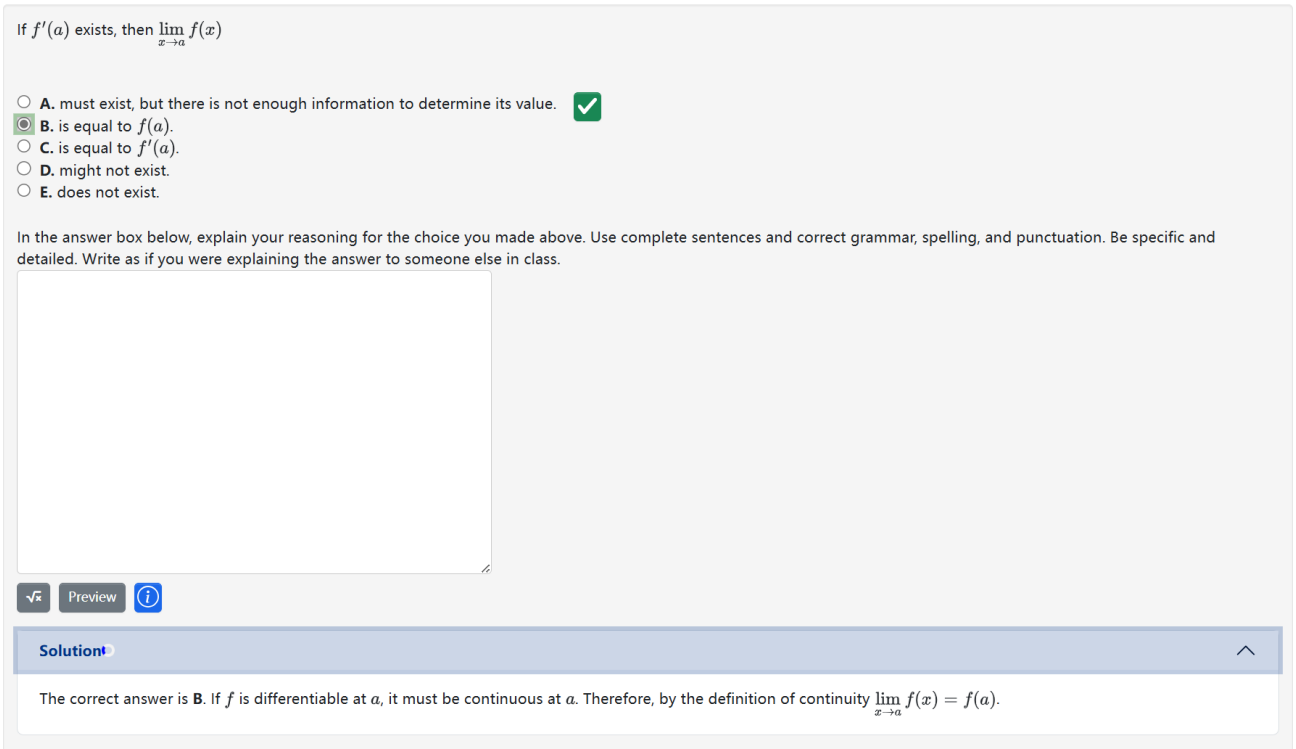
- ✧ Automatic grading and instant feedback
- ✧ Random parameters
- ✧ Support Latex mathematical notation
- ✧ Question library (more than 10,000 predefined questions)
  GitHub - openwebwork/webwork-open-problem-library: A library of WeBWorK problems contributed by the OpenWeBWorK community
- ✧ Question authoring

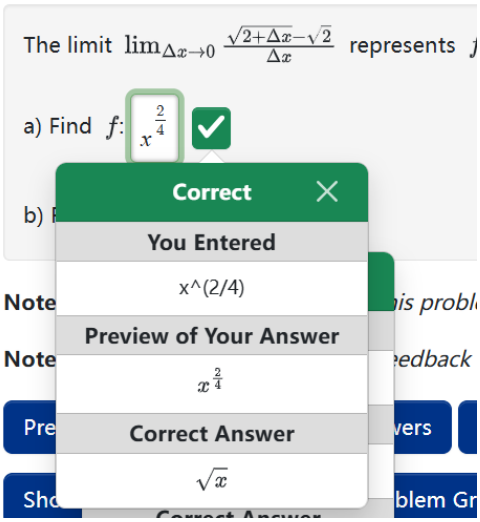
3. Question types supported

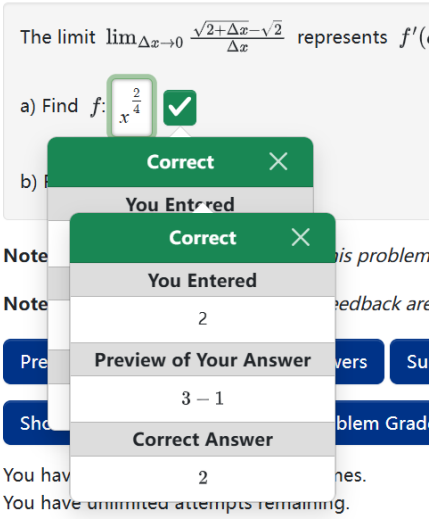
- ➢ **数值计算题**：如数学运算、物理公式计算等，系统可自动验证答案。
- ➢ **符号表达式题**：允许输入数学符号或公式（如积分、微分方程）。
- ➢ **图像题**：例如绘制函数图像或几何图形，系统可识别关键点或形状。
- ➢ **多选题/填空题**：有限选项的题目，支持随机参数生成不同变体。

*** 开放式题目：**

WeBWorK 无法自动批改开放式主观题（如论述题、证明题、开放式写作），这类题目需要人工评分。系统依赖预设的正确答案或计算规则，无法处理自由文本的逻辑分析。



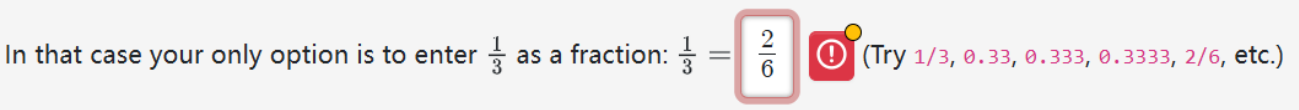




*** WebWork 批改时的神奇特性：** [WeBWorK : TEST]

a) Decimal Approximations:

Sometimes, your instructor will require you to enter an answer **exactly**.



* Only $\frac{1}{3}$ works, the others all fail.

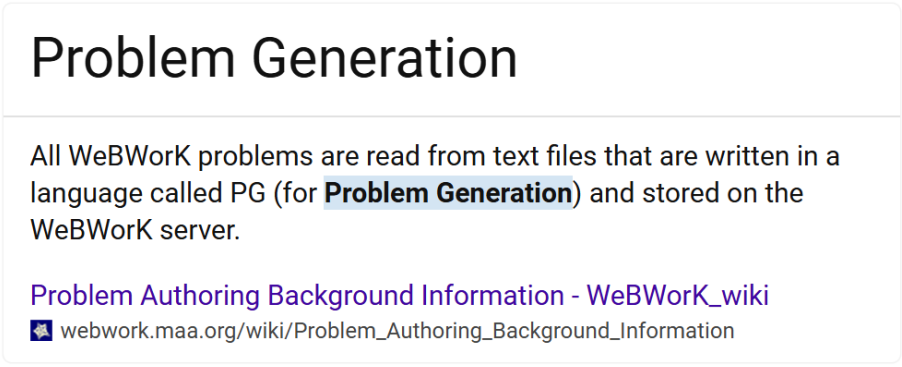
Sometimes, you will be allowed to use **decimal approximations** to the real answer.

Why's that? If a decimal approximation is acceptable at all, then you need to use enough significant digits so your answer is "close enough" to the actual correct answer (which was $1/3$ in this case). In general, using **four significant digits** in your decimals will be enough. You will often be able to get away with fewer, but using four is recommended.

Examples:

* $\sqrt{110}\approx 10.48809$, $\frac{1}{491}\approx 0.00203666$

Decimal tolerance settings may vary from problem to problem and the above is only describing **default decimal tolerance**. For example, if a problem has a **monetary answer**, it might expect you to answer correctly all the way to the **hundredths place**, even if it is a large amount in thousands of dollars. Watch out for any **specific instructions** in each problem that tell you how precise you should be.



基于 Perl，与 LaTeX 集成。

Related links:

WeBWorK:
https://github.com/openwebwork

Authors - WeBWorK_wiki
--Introduction to PGML - WeBWorK_wiki
--PGLabs - WeBWorK_wiki

the_structure_of_a_webwork_problem [WeBWork Wiki]
--Basic Perl syntax - WeBWorK_wiki
--Randomizing parameters in a WeBWorK Problem

GitHub - openwebwork/pg: Problem rendering engine for WeBWorK

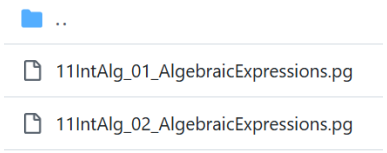
Learning How to Author Problems - WeBWorK_wiki
--SampleProblem1 - WeBWorK_wiki
--……

Categories - WeBWorK_wiki

Youtube:

John Travis - YouTube
--WeBWorK Problem Authoring - Getting Started Basics - YouTube

To start:



A question in WeBWorK is written as a **PG file (xxx.pg)**, which consists of **five sections**:

1. A ***tagging and description section***, that describes the problem for future users and authors, (开头注释)
2. An ***initialization section***, that loads required macros for the problem, (初始化加载宏)
3. A ***problem set-up section*** that sets variables specific to the problem, (定义问题中所使用的变量)
4. A ***text section***, that gives the text that is shown to the student, and (问题描述文本)
5. An ***answer and solution section***, that specifies how the answer(s) to the problem is(are) marked for correctness, and gives a solution that may be shown to the student after the problem set is complete. (答案结果展示)

Sample Problem 1 (to show basic structure):

PG codes	Explanation

`# 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')`
	Tagging and description section
	
	Any line that begins with a **"#"** is a ***comment*** for other authors who read the problem, and is not interpreted by !WeBWorK.
	
	All of the tagging information exists to allow the problem to be **easily indexed**. Because this is a sample problem there isn't a textbook per se, and we've used some **default tagging values**. There is an on-line list of current chapter and section names and a similar list of keywords, as well as a page of best practices for tagging problems. The list of **keywords** should be **comma separated** and **quoted** (e.g., `KEYWORDS('calculus','derivatives')`).

`DOCUMENT();`
`loadMacros(`
`"PGstandard.pl",`
`"MathObjects.pl",`
`"PGcourse.pl",`
`);`
	Initialization section
	The **first executed line** of the problem **must** be the `DOCUMENT();` command. Note that every command ***must end with a semicolon***.
	
	The **loadMacros** command loads information that works behind the scenes. For our purposes we can usually just load the macros shown here and not worry about things further.
	必须首先加载 PGstandard.pl，再加载其他宏包（如 MathObjects.pl 依赖基础环境）。

`# make sure we're in the context we want`
`Context("Numeric");`
 	
`$a = random(2,9,1);`
`$trigFunc = Formula("sin($a x)");`
`$trigDeriv = $trigFunc->D();`
	Problem set-up section (What we Generate)
	
	`Context("Numeric");` sets the **"context"**, which **determines how variables are interpreted**. Contexts and context explanations are given on this help page.
	
	The bulk of the set-up section defines variables that we use in the rest of the problem.
	
	All *scalar variables* are prefaced with a **dollar sign**: thus `$a` is a **variable** that has a **(non-vector, non-array) value**. We also define `$trigFunc` to be a MathObject Formula, which means that it knows things about itself, in particular, **how to find its own derivative**, which we find with the expression `$trigFunc->D()`.

`TEXT(beginproblem());`
`Context()->texStrings; ?`
`BEGIN_TEXT`
`Find the derivative of the function \(f(x) = $trigFunc\).`
`$PAR`
`\(\frac{df}{dx} = \) \{ ans_rule(35) \}`
`END_TEXT`
`Context()->normalStrings;`
	Text section
	
	`TEXT(beginproblem());` line displays a **header** for the problem;
	
	`Context()->texStrings;` line sets **how formulas are displayed** in the text, and we reset this after the text section.
	
	Everything between `BEGIN_TEXT` and `END_TEXT` lines (each of which must appear alone on a line) is **shown to the student**.
	
	Mathematical equations are delimited by `\( \)` (for inline equations) or `\[ \]` (for displayed equations); in these contexts inserted text is assumed to be TeX code.
	There are a number of variables that set formatting: `$PAR` is a **paragraph break** (like **\par** in **TeX**). This page gives a list of variables like this. Finally, `\{ \}` sets off *code that will* ***be executed in the problem text***.
	Here, `ans_rule(35)` is a function that inserts an **answer blank** 35 characters wide.

`ANS( $trigDeriv->cmp() );`
 	
`Context()->texStrings;`
`BEGIN_SOLUTION`
`We find the derivative to this using the chain rule. The inside function is \($a x\),`
`so that its derivative is \($a\), and the outside function is \(\sin(x)\), which has derivative \(\cos(x)\). Thus the solution is`
`\[ \frac{d}{dx} $trigFunc = $trigDeriv. \]`
`END_SOLUTION`
`Context()->normalStrings;`
 	
`ENDDOCUMENT();`
	Answer and solution section **(What we Generate)**
	
	The problem answer is set by the `ANS( $trigDeriv->cmp() );` line, which simply says that the answer is marked by **comparing the student's answer** with the trigonometric function derivative that we defined before.
	
	Then, we explain the solution to the student. This solution will show up when the student clicks **"show solution"** after they've finished the problem set.
	
	The `ENDDOCUMENT();` command is the last command in the file.

Formal specification:
Contact Us - WeBWorK_wiki
Mailing Lists - WeBWorK_wiki

Parser

- We have learned CFGs from our previous lecture.
- Now we aim to develop a parser that will
  - takes a sequence of tokens as input;
  - analyze the syntactic structure of that input;
  - generates a corresponding parse tree; and
  - Identify the syntax errors if there is any.
- There are two types of parsing: *top-down* and *bottom-up*.
  - Top-down parsing: **start** from the root of the parse tree, work down its way, attempting to match the input tokens with leaf nodes.
  - Bottom-up parsing: start from lowest level of parse tree with individual tokens, and work up by the rules of grammar and reduce the tokens to the start symbol. (reverse rightmost derivation)
- Top-down parsing is only for a subset of CFGs, it may not be able to handle complex grammars effectively.