Московский государственный университет имени М. В. Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра алгоритмических языков

Н. К. Маркитантова

Генерация однотипных математических задач по заданному пользователем шаблону

Курсовая работа

Научный руководитель
к.ф.-м.н., доцент А. В. Столяров

Москва, 2016

Содержание

1 Введение

Математика – один из основных предметов школьной программы, а так же профильный экзамен при поступлении на многие специальности. При изучении этого предмета и подготовке к экзаменам важно «набить руку» в решении задач, что легко достигается при решении множества однотипных заданий. Более того, многие согласятся, что наличие на контрольной или самостоятельной работе у каждого ученика индивидуального варианта сильнее мотивирует их к изучению материала, так как нет никакой возможности списать у одноклассников. Как для подготовки, так и для таких проверочных работ необходимо иметь довольно обширную базу аналогичных заданий. Однако составить такую подборку – довольно объемная работа.

Под однотипными понимаются задачи, различающиеся только числами и второстепенными деталями условия. Их можно задать шаблоном и правилами генерации изменяемых значений. Шаблон в данном случае – условие задачи с использованием специальных символов там, где должны находиться переменные величины. Правила генерации изменяемых значений – решение прямой или обратной задачи, записанное на каком-либо алгоритмически полном языке.

При данном подходе человеку, задающему шаблон, необходимо уметь легко решать прямые и обратные задачи и рационально выбирать какой из приемов в каждом частном случае эффективне. Например, для квадратного уравнения гораздо проще решить обратную задачу и сначала сгенерировать ответы, а затем с помощью теоремы Виета найти коэффициенты. В случае же задачи дифференцирования лучше применять «прямой» подход, так как задача интегрирования в символном виде сложнее в реализации.

Настоящая работа посвящена созданию программы, генерирурующей описанные однотипные математические задачи по заданному шаблону. Шаблон, как и количество задач, должны задаваться пользователем с помощью конфигурационного файла и параметров командной строки. Результатом программы должны быть два файла в формате pdf: один с условием задач, а другой с ответами к ним. Важно, чтобы внутри одного файла задания не повторялись.

Основным языком описанной программы является C++. Для задания и вычисления правил генерации изменяемых значений внутри конфигурационного файла, а так же для решения некоторых частных задач внутри программы используется InteLib – библиотека классов на C++, позволяющая программировать в стиле функционального языка Lisp в рамках обыкновенной программы без какой-либо дополнительной обработки.

2 Программная реализация

В этой главе рассматривается внутреннее устройство программы, основные классы и используемые библиотеки.

2.1 Краткое описание алгоритма

На вход программе подаётся конфигурационный файл, описывающий задачу и число, обозначающее сколько задач должно быть сгенерировано. Конфигурационный файл, состоящий из 4 частей:

1. Шаблон условия задачи;
2. Описание констант;
3. Описание алгоритма генерации изменяемых значений;
4. Шаблон ответа к задаче.

Следующее кратко описывает алгоритм работы программы:

1. Создаются файлы `«task.tex»` и `«answer.tex»`, добавляются в них преамбулы для последующего создания pdf-файлов;
2. Конфигурационный файл разбирается на отдельные лексемы следующих типов: `string` – строка, `variable` – переменная, `number` – число, `sign` – знак «+». Переменная – набор символов латинского алфавита и нижнее подчёркивание, идуще после специального символа «%», число – произвольная последовательность цифр, составляющая целое или вещественное число, знак «+» – символ «+», идущий сразу после специального символа «%», строка – произвольный набор символов, не являющийся частью других лексем. Создаётся 4 отдельных списка лексем – по одному на каждую часть конфигурационного файла;
3. По списку лексем первой и второй части конфигурационного файла строится таблица переменных;
4. Построенная таблица передаётся в `IntelibReader` – объект из библиотеки InteLib, позволяющий считывать и выполнять Lisp-подобные выражения;
5. Лексемы из третьей части конфигурационного файла передаются в уже настроенный `IntelibReader`, который вычисляет эти выражения;
6. Вычисленные значения вместе с шаблоном записываются в уже подготовленные в п.1 файлы;
7. Последние 2 пункта выполняются ровно по количеству задач, которые надо сгенерировать;
8. Из построенных файлов с помощью вызова вспомогательной программы `pdflatex` создаются pdf-файлы.

2.2 Разбор конфигурационного файла

Основную часть работы составляет разбор конфигурационного файла на лексемы, так как он состоит из нескольких частей, каждая из которых имеет собственную семантику. В силу слабых ограничений на порядок следования лексем в каждой из частей в настоящее время реализован только лексический анализ. Любая семантическая и синтаксическая ошибка может привести к сбою на других этапах генерации, или же результат работы программы будет отличаться от ожидаемого.

Всего построено 3 лексических анализатора (так как первая и последняя часть конфигурационного файла имеют одинаковую структуру). Все они являются наследниками общего абстрактного класса `LA` и различаются только автоматами, которые они реализуют.

Для лексического анализатора первой и последней шаблонных частей конфигурационного файла используется класс `LA_template`. На выходе из него получается список, состоящий из лексем всех возможных типов.

Для лексического анализатора второй части используется класс `LA_attribute`. На выходе из него получается список из лексем типа `variable` и `number`.

Для лексического аназитора части вычисления используется класс `LA_solve`. На выходе из него получаем единственную лексему – строку, содержащую все символы внутри блока.

Таблица переменных задается структурой `Table` и составляется на основе списка лексем первой и второй части конфигурационного файла.

2.3 Работа с библиотекой InteLib

При реализации программы была использована библиотека Intelib, а именно класс `IntelibReader`, который позволяет считывать и выполнять Lisp-выражения.

Для того, чтобы объект `IntelibReader` смог обрабатывать выражения, необходимо задать ему, какие символы являются специальными, то есть названиями переменных, констант и функций. В данной реализации задано всего 6 функций: присваивание, сложение, вычетание, умножение, деление и генерация случайного значения из заданного диапазона. Все функции, кроме последней, были уже реализованы внутри библитеки для класса `LispReader`, использующегося в интерпретаторе IntelibLisp. В данном экземпляре класса `IntelibReader` использовались они же.

Для последней функции генерации случайного значения из заданного диапазона реализован класс `LFunctionGenerate`, являющийся наследником встроенного класса `SExpressionFunction` и релизующий необходимую функцию.

Строка, считанная в процессе разбора третьей части конфигурационного файла, загружается в объект класса `IntelibReader` с помощью метода `FeedString()`, получающего на вход строку символов. Объект заданного класса формирует из строки InteLib-выражения, которые можно «достать» из него с помощью метода `Get()`. Все полученные выражения вычисляются, вследствие чего формируется новая таблица переменных, но уже внутри объекта класса `IntelibReader`.

При необходимости получить значение переменной тоже используется объект класса `IntelibReader`. Аналогично вычислению выражений, ему подаётся на вход имя необходимой переменной, затем «достаётся» вычисляемое Intelib-выражение, которое можно представить в виде строки или числа и вывести на экран.

3 Руководство пользователя

В этой главе рассматриваются команды для запуска реализованной программы, а так же подробно разобрана семантика конфигурационного файла с примерами.

3.1 Запуск программы

Для установки необходимо зайти в корневую папку, в которой располагается программа. Затем в командной строке выполнить команду `make all`. Далее можем запустить программу из терминала командой

```
./task_generator FILENAME N
```

Где:

- `FILENAME` – название конфигурационного файла с расширением, с указанием относительного пути от корневой папки программы;
- `N` – количество задач, которое необходимо сгенерировать.

3.2 Конфигурационный файл

Конфигурационный файл делится на 4 части – директивы, каждая из которых выполняет собственную функцию и имеет собственную семантику. Директивы внутри конфигурационного файла отделяются друг от друга символами `%%%`, написанными на отдельной строке между блоками. Сразу после отделяющих символов и до окончания строки может идти название директивы или иной комментарий. Все директивы являются обязательными и находятся в строго заданном порядке, однако могут иметь пустое тело.

Краткое описание директив в требуемом порядке указания в конфигурационном файле:

- `TEMPLATE` – шаблон условия задачи;
- `ATTRIBUTE` – задание констант, необходимых при вычислениях;
- `SOLVE` – задание алгоритма генерации и вычисления изменяемых значений в условии;
- `ANSWER` – шаблон ответа к задаче.

Рассмотрим эти директивы и их семантику подробнее.

3.2.1 Директива `TEMPLATE`

Первая директива конфигурационного файла, которая позволяет задать шаблон условия генерируемых задач. Шаблон задается при помощи языка разметки LaTeX. В настоящей работе его семантика приводиться не будет, желающие могут ознакомиться с ним в пособии [1].

Отличием от обычного LaTeX-фрагмента является наличие изменяемых частей – переменных. Переменная может состоять из произвольного набора латинских символов и символа нижнего подчеркивания. В тексте шаблона для обозначения переменной используется специальный символ «%», сразу после него без разделителей указывается имя переменной. Если после специального символа «%» находится разделитель или любой другой символ, не являющийся частью названия переменной, то программа распознаёт специальный символ как символ процента.

Особенностью обозначения переменных является то, что иногда необходимо вывести знак этой переменной, даже если она положительна. Чтобы переменная в любом случае выводилась со знаком, необходимо сразу после специального символа «%» использовать еще один специальный символ – «+», после которого без разделителей указывается имя переменной.

Рассмотрим пример директивы `TEMPLATE`:

```
%%%TEMPLATE
Решите квадратное уравнение: $$ %a x^2 %+b x %+c = 0 $$
```

В рассмотренном примере:

- `%%%` – обозначение начала новой директивы;
- `TEMPLATE` – комментарий к директиве, в данном случае обозначающий её название;
- `a` – числовая переменная, выводится без знака;
- `b` – числовая переменная, выводится со знаком в любом случае (необходимо, чтобы выражение в шаблоне имело вид выражения и не приходилось рассматривать несколько случаев шаблона);
- `c` – числовая переменная, выводится со знаком в любом случае;
- всё остальное – неизменяемая часть шаблона условия задачи.

3.2.2 Директива `ATTRIBUTE`

Вторая директива конфигурационного файла, которая позволяет задавать значения констант, необходимых для дальнейшего вычисления и генерации изменяемых значений. Все выражения внутри этой части имеют вид:

```
<ИМЯ_ПЕРЕМЕННОЙ> = <ЗНАЧЕНИЕ_ПЕРЕМЕННОЙ>
```

Где

- `ИМЯ_ПЕРЕМЕННОЙ` – набор заглавных символов латинского алфавита и знака нижнего подчеркивания;
- `ЗНАЧЕНИЕ_ПЕРЕМЕННОЙ` – целое или вещественное число.

Рассмотрим пример директивы `ATTRIBUTE`:

```
%%%ATTRIBUTE

MAX_ANS = 50
MIN_ANS = 2
```

В рассмотренном примере:

- `%%%` – обозначение начала новой директивы;
- `ATTRIBUTE` – комментарий к директиве, в данном случае обозначающий её название;
- `MAX_ANS, MIN_ANS` – названия констант;
- `50, 2` – значения соответствующих констант.

3.2.3 Директива SOLVE

Третья директива конфигурационного файла, которая позволяет генерировать и вычислять изменяемые части – переменные. Директива имеет Lisp-подобную семантику, а именно каждое выражение заключается в скобки, где на первом месте стоит функция, а затем её аргументы. Подробнее о языке программирования Lisp и его семантике можно почитать в пособии [2].

Директива состоит из набора выражений, где выражение задаётся следующим образом:

```
<ВЫРАЖЕНИЕ> ::= <ЧИСЛО> | <ПЕРЕМЕННАЯ> | <КОНСТАНТА> | (<ИМЯ_ФУНКЦИИ> [<АРГУМЕНТ>])
```

Пользователю доступно всего несколько функций:

- `(= <ПЕРЕМЕННАЯ> <ВЫРАЖЕНИЕ>)` – присваивание. Присваивает переменной значение выражения;
- `(+ <ВЫРАЖЕНИЕ> <ВЫРАЖЕНИЕ>)` – сумма значений двух выражений;
- `(- <ВЫРАЖЕНИЕ> <ВЫРАЖЕНИЕ>)` – разность значений двух выражений (из первого вычетается второе);
- `(* <ВЫРАЖЕНИЕ> <ВЫРАЖЕНИЕ>)` – произведение значений двух выражений;
- `(/ <ВЫРАЖЕНИЕ> <ВЫРАЖЕНИЕ>)` – частное значений двух выражений (первое делится на второе);
- `(GENERATE MIN MAX)` – генерирует случайное число в диапазоне `[MIN, MAX)`, где `MIN` и `MAX` являются выражениями.

Чтобы задать новую переменную, пользователю достаточно присвоить ей какое-либо значение.

Рассмотрим пример директивы `SOLVE` для генерации коэффициентов квадратных уранений и ответов к полученным выражениям. На самом деле проще решить обратную задачу и сгенерировать ответы в заданном диапазоне, а по ним по теореме Виета найти коэффициенты:

```
%%%SOLVE

(= x1 (generate -5 5))
(= x2 (generate -5 5))
(= a (generate 2 5))
(= b (- (* a (+ x1 x2))))
(= c (* a (* x1 x2)))
```

В рассмотренном примере:

- `%%%` – обозначение начала новой директивы;
- `SOLVE` – комментарий к директиве, в данном случае обозначающий её название;
- `a, b, c` – коэффициенты квадратного уравнения;
- `x1, x2` – корни соответствующего квадратного уравнения.

3.2.4 Директива ANSWER

Четвёртая и последняя директива конфигурационного файла, которая позволяет задать шаблон ответа к задаче. Семантика этой части полностью совпадает с семантикой директивы TEMPLATE, за исключением того, что все имена переменных, используемых в условии, должны быть уже заданы и/или вычислены в предыдущих директивах.

Рассмотрим пример директивы ANSWER:

```
%%%ANSWER
$x_1 = %x1, x_2 = %x2$
```

В рассмотренном примере:

- %%% – обозначение начала новой директивы;
- ANSWER – комментарий к директиве, в данном случае обозначающий её название;
- x1, x2 – названия переменных, вычисленных в предыдущих директивах;
- всё остальное – неизменяемая часть шаблона ответа к задаче.

4 Заключение

4.1 Результаты

В рамках курсовой работы было сделано:

- Создана демонстрационная версия программы, способная генерировать математические задачи по шаблону с неизменяемым текстом, исключительно вещественными значениями изменяемых величин и элементарными операциями для их генерации (сложение, вычетание, умножение, деление и выбор случайного значения переменной из заданного диапазона);
- Созданы несколько примеров конфигурационных файлов с шаблонами элементарных математических задач, удовлетворящих заданным ограничениям;
- Проведено тестирование на описанных конфигурационных файлах.

4.2 Перспективы

В дальнейшем планируется:

- Дополнить систему возможностью работы с более сложными операциями: дифференцированием, возведением в степень, вычислением логарифма, а так же условными выражениями;
- Реализовать типизацию переменных, добавить иные типы переменных, кроме числовых;
- Реализовать информативную диагностику ошибок;
- Реализовать «красивый» вывод выражений: убрать незначащие цифр, не выводить единичные коэффициенты;
- Дополнить конфигурационный файл новой директивой, с помощью которой пользователь сможет вводить собственные функции на языке Lisp.

Список литературы

[1] А. В. Столяров. *Сверстай диплом красиво: LaTeX за три дня.* Изд-во МАКС Пресс, 2010.

[2] М. Ю. Семенов. *Язык лисп для персональных ЭВМ.* — М.: Изд- во Моск. ун-та, 1989.