Balancing chemical equations

Andrea Princic

July 22, 2022

Balancing chemical equations

Report about the second Learning Unit for the course of Methods in Computer Science Education: Analysis.

Class

The Learning Unit is designed for students in the **fourth year of high school** (Liceo Scientifico).

Interdisciplinary objectives and motivations

The goal of this LU is to teach students how to balance a chemical equation, a topic in **Chemistry**. To do this they will need basic knowledge about atoms, table of elements and chemical notation. They will also need to know how equations work in general and how to manipulate them.

Programming objectives and motivations

With respect to programming, the main goal is to teach students the basic concepts of **parsing formulas** (mainly using the material given by the professor) and how to apply and manipulate **mathematical operations**.
This will be done in the context of a mobile application using the programming language **Java Script**, which will also require a minimal usage of event-driven programming.
Prerequisites in Java Script are:

- variables
- conditionals
- loops
- functions
- string manipulation
- associative arrays

Students will learn how to **handle events** in a mobile application using buttons and text fields. They will have to create an algorithm that solves a chemical equation with the given parameters.

In addition to the equation task, students can also implement an extra task of their choice, to further explore the possibilities of mobile applications.

Materials

Students will be given a **set of functions** useful to parse the equation.
Only one function is needed to be directly used by the students, but they are invited to look into all of them to **better understand** how they work.
All functions are preceded by an accurate description of their parameters and return value, including some examples.
The code inside is written in a way that is **easy to understand** for everyone: variables are declared just before they are used, so students won't need to go up and down the code to find the declaration, and every name is meaningful and descriptive. This will come in handy for students with **dyslexia**.

These functions will only parse a **correctly formatted** equation, meaning that students can, if they want, try to filter out incorrectly formatted equations using regexes. Doing so will give them an extra point.

Delivery

The project will be delivered as a **homework** with a two-week deadline.

Evaluation

Criterion	**Max score**
Correct implementation of the balancing algorithm	30
Correct use of event-driven programming	20
Maximum value supported for coefficients (must be at least 6)	20
Maximum number of terms supported (must be at least 4)	20
Code quality	10
Additional task (extra)	10
Equation validation (extra)	10

Scores are scaled to 0–10.

Correct implementation of the balancing algorithm

The most important criterion is the correct implementation of the balancing algorithm. The correctness of the algorithm will be tested on the following test cases:

Equation	**Solution**
$NH_3 + O_2 \rightarrow N_2 + H_2O$	4, 3, 2, 6
$H_2 + O_2 \rightarrow H_2O$	2, 1, 2
$P_4O_{10} + H_2O \rightarrow H_3PO_4$	1, 6, 4
$CO_2 + H_2O \rightarrow C_6H_{12}O_6 + O_2$	6, 6, 1, 6
$Al + HCl \rightarrow AlCl_3 + H_2$	2, 6, 2, 3
$2Na_2CO_3 + HCl \rightarrow NaCl + H_2O + 2CO_2$	1, 4, 4, 2, 1
$2SiCl_4 + H_2O \rightarrow H_4SiO_4 + HCl$	1, 8, 2, 8
$C_7H_6O_2 + O_2 \rightarrow CO_2 + H_2O$	2, 15, 14, 6

The first five tests are needed to get the sufficient score: they all have a maximum value of 6 for coefficients and at most 4 terms.

Use of event-driven programming

Event-driven programming is needed to interact with the **Graphical User Interface**. Students are required to use the proper functions to add **handlers** that let them receive input from the user and display some output.

Maximum coefficient and terms

The balancing algorithm must at least be able to solve equations that have a maximum value of 6 for coefficients and at most 4 terms.
Writing an algorithm that can handle equations with **unlimited terms and coefficient value** will grant the student the maximum score, although is permitted (and suggested) to limit the maximum value of coefficients via some kind of input to **avoid expensive computation** when solving equations with large coefficients.

Code quality

Code quality will be evaluated based on the following criteria:

- number and utility of **comments**
- variables and functions name's **clarity and descriptiveness**
- code organization (**functions**)
- generalization and abstraction

The GUI quality and simplicity is also considered here.

Extras

Students can implement some **extra task**, if they want.
One is to **validate the equation**, and the other is to **implement a new feature** for the application. The validation can be done using a **regex**, taking inspiration from the one already implemented in the code if they need. The new feature can be anything related to the topic: atoms, table of elements, equations.
This will let them explore the possibilities of mobile applications and challenge their **creativity**.

Solutions

Best solution

The 10/10 solution presents a perfect algorithm that can **solve any equation**, regardless of the number of terms and coefficient value.
It allows to compute coefficients for any number of terms with any given limit in a smart way: it treats coefficients as a number that goes from 1 to $limit^{terms}$ and then divides it many times to extract all the singular coefficients. This algorithm is very well **generalized and scalable**.
The maximum coefficient value is limited to 15 though, to avoid expensive computation. This is completely fine and also needed, otherwise the algorithm could never end.
The interface is good to see and **easy to use**, with a slider that lets you choose the maximum coefficient value.
Code is fully commented and **self-descriptive**, with clear variables and functions names that allow the reader to understand what the code is doing without any effort. Functions are well organized in **simple tasks**, making the flow clear and easy to follow.
The GUI and the logic are **completely separated** from each other and the communication between them is one-directional, as it should be.

The student even implemented both the additional tasks, (partially) **validating the equation** and adding the feature to **read information** about every entry in the table of elements.

Sufficient solution

The 6/10 solution proposes a working algorithm that can only solve **simple equations**: at most 4 terms and a maximum (fixed) value of 6 for coefficients.

The maximum coefficient is **not customizable**, which in some cases may cause a waste of time if the user already knows that it won't exceed a certain limit.
The interface is **very essential**, with just a text area and a button to start the computation.
Code is **completely uncommented**, and variables and functions names are not very descriptive (mainly **single letters**).
Especially the algorithm is **not generalized** at all and definitely **not scalable**, considering the limit hard-coded inside loops and the way coefficients are generated for each singular case (2, 3 or 4).