Spreadsheets in Education (eJSiE)

Volume 1 | Issue 1 Article 3

July 2003

On Programming and Spreadsheet Calculations

Gabriela Lovászová
Univerzita Konštantína Filozofa, glovaszova@ukf.sk

Jozef Hvorecký
Vysoká škola manazmentu, jhvorecky@vsm.sk

Recommended Citation

Lovászová, Gabriela and Hvorecký, Jozef (2003) On Programming and Spreadsheet Calculations, *Spreadsheets in Education (eJSiE)*: Vol. 1: Iss. 1, Article 3.
Available at: http://epublications.bond.edu.au/ejsie/vol1/iss1/3

On Programming and Spreadsheet Calculations

Abstract
"Experiment", "discovery" and "self-teaching" are basic components of active learning. Despite the fact that learners are investigating notions and relationships known long before, their own discovery of something new and unknown for them results in their fascination with an irreplaceable educational value. Spreadsheets are one of many environments that can successfully be used for building stages for similar experimentation.

Activities described in this paper show how spreadsheet-based calculations can lead students to a deeper comprehension of algorithms, their execution and notation in a programming language. Their goal is to identify a relationship between a notation of an algorithm and the series of figures produced by its execution. During their investigations, students exploit the adaptability of spreadsheets–each change of an input value evokes the complete recalculation of the entire sheet. The instant recalculations allow the students to observe many "runs" of the studied algorithm, to formulate hypotheses and to verify them much faster than any other methods of traditional programming.

Keywords
Introductory programming courses, constructive learning, simulation of program tracing

On Programming and Spreadsheet Calculations

Gabriela Lovászová
Univerzita Konštantína Filozofa
949 74 Nitra, Slovakia
glovasz@ukf.sk

Jozef Hvorecký
Vysoká škola manažmentu
820 09 Bratislava, Slovakia
hvorecky@cutn.sk

July 24, 2003

Submitted March 2003; revised and accepted June 2003.

Abstract

"Experiment", "discovery" and "self-teaching" are basic components of active learning. Despite the fact that learners are investigating notions and relationships known long before, their own discovery of something new and unknown for them results in their fascination with an irreplaceable educational value. Spreadsheets are one of many environments that can successfully be used for building stages for similar experimentation.

Activities described in this paper show how spreadsheet-based calculations can lead students to a deeper comprehension of algorithms, their execution and notation in a programming language. Their goal is to identify a relationship between a notation of an algorithm and the series of figures produced by its execution. During their investigations, students exploit the adaptability of spreadsheets—each change of an input value evokes the complete recalculation of the entire sheet. The instant recalculations allow the students to observe many "runs" of the studied algorithm, to formulate hypotheses and to verify them much faster than any other methods of traditional programming.

Keywords: Introductory programming courses, constructive learning, simulation of program tracing.

1 Introduction

Results of international high-school competitions in Mathematics, Computer Science and other science-related disciplines indicate that education in Eastern and Central Europe has been of a high quality. At the same time, these subjects are not popular among youngsters and many of them experience problems with them at high schools and universities. These students describe the subjects as being difficult to understand, with no relationship to their lives and boring [26].

This issue seems to originate from universities. As a rule, future teachers are trained in two related subjects. For example, at the University of Constantinus the Philosopher in Nitra, Computer Science is studied in a combination with Mathematics or Physics or English. Traditionally, science-related courses for future researchers and pre-service teachers have been almost identical. They are very intensive and (for all teachers above Grade 4) lead to an equivalent MSc. title. Typical teaching methodologies stress formal knowledge and its exact usage. They are minimally oriented on raising students' interest and motivation. Such an approach is quite natural as university students are in general interested in their study fields. Teachers tend to simulate the behaviour of their educators. As a result many of them later repeat "lecturing" in primary and secondary schools. The result has been mentioned: Gifted pupils and students are gaining good knowledge and excel in Olympiads. Those less involved are becoming lost and step-by-step start ignoring the course. This results in a low average level of mathematics and science knowledge among the population.

There have been isolated attempts to change the model, e.g. [9]. However, the absence of a "critical mass" of supporters results in little or no change, even after more than a decade of the collapse of the former regime. The reasons are obvious—the instant replacement of teachers is impossible and the substantial change of their in-class behaviour requires a long time. We believe that teacher preparation with a different orientation is the best long-term strategy—it would accelerate the building of a critical mass of innovators. In our courses, we demonstrate the methods of active learning and train our students in their use. They also develop their own methods as a part of their master thesis (e.g. [18], [17]). Our experience and discussions with our graduates show that they have started using various approaches in their classes.

In this article we describe one of the ways—the use of spreadsheets for demonstrating relationships between programs and their executions. The problem set is oriented to Pascal and used at the University of Constantinus the Philosopher in Nitra, Slovakia, for training future Computer Science teachers. It is important to add that the described training is run in one of the final courses. The participants are already skilled programmers. Thus, its main aim is explaining alternative methods to teaching programming and introducing various tools for demonstrating its concepts (including the traditional ones as flowcharts, debuggers, algorithm animators, etc.) Generic software (spreadsheet programs, in particular) is among these concepts because it offers a cheap way to create a wide variety of different learning environments for both mathematics and computer science. In recent years, a set of similar "intellectual exercises" have been presented ([11], [12], [20], [21]). It is important to add that there is another approach to introductory programming courses in Slovakia based on minilanguages ([3], [16]).

2 Making learning active

The discussed methodology best coincides with constructive learning [2].

1. Set the stage but have the students generate the knowledge for themselves as much as possible [13], [1].

2. Anchor the knowledge in authentic situations and activities [4].

3. Use the cognitive apprenticeship methods of modeling, scaffolding, fading and coaching to convey how to construct knowledge in authentic situations and activities [5].

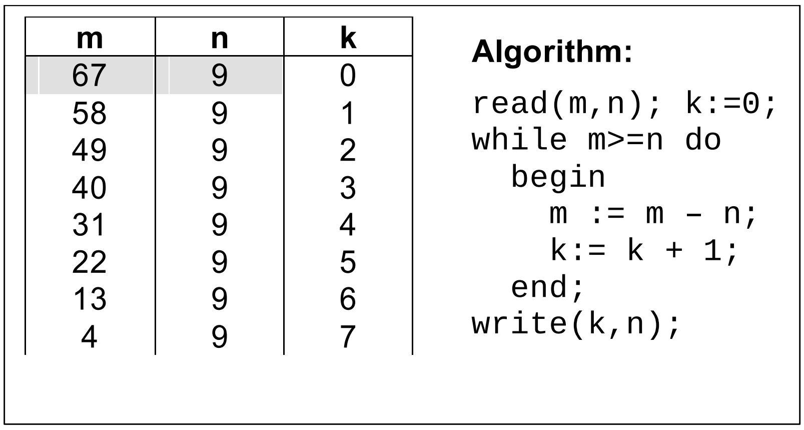
4. Situate knowledge in multiple contexts to prepare for appropriate transfer to new contexts [8].

5. Create cognitive flexibility by ensuring that all knowledge is seen from multiple perspectives [25].

6. Have the students collaborate in knowledge construction [14].

Teaching programming is very appropriate for implementing the above ideas. The fact that teaching programming cannot be done without actively working on computers is the first positive step. Unfortunately, it is too often the last one, too. To make the next steps, the teaching methodology can also exploit other computerized tools [15], [23]. We are investigating the potential of spreadsheet programs for such a support.

When spreadsheets first appeared, they were welcomed as "a success story of making programming easier" [19]. Three years later, Nardi and Miller [24] added that the biggest advantage of spreadsheets is not cognitive but motivational: "...after only a few hours of work, spreadsheet users are rewarded by simple but functioning programs". There are good reasons for such claims as spreadsheet programmers, for example, do not need to tackle (sometimes very peculiar) input/output commands.



m	n	k
67	9	0
58	9	1
49	9	2
40	9	3
31	9	4
22	9	5
13	9	6
4	9	7

Algorithm:

```
read(m,n); k:=0;
while m>=n do
  begin
    m := m - n;
    k:= k + 1;
  end;
write(k,n);
```

Figure 1: The notation and the results of an algorithm.

Notice that these opinions refer to the period when computer literacy was assumed to be identical to programming and for that reason programming languages were taught more frequently than other software tools. Our present situation is different. Most computer literacy courses target text processing, computer graphics and spreadsheet calculations. As a result, teaching programming can profit from the prerequisite. In this paper we are looking for potential ways of its exploitation: presuming that students are familiar with spreadsheets, we build environments in which they can more easily understand programming concepts. Microsoft Excel [7], [6] was used in the following examples.

The general strategy for setting up the stage can be described as: **Programming problems can be also solved using spreadsheet calculations. The stage is built around one of the spreadsheet solutions. It is selected and arranged in a way that exhibits and amplifies the considered properties of the algorithm. By experimenting with the spreadsheet solution, the students reveal the properties and extrapolate them to the field of programming.**

Examples in [10] illustrate relationships between spreadsheet formulas and programming language commands, show reasons for setting up initial values of variables, demonstrate a way of "translation" of a sequence of formulas into a loop, and introduce the idea of computational complexity. In this paper, we simulate executions of algorithms written in a particular language (Pascal) and ask students to identify their properties.

3 Setting up the stage

As we concentrate on the relationship between a notation and its execution, all other (pointless) features of the problem are removed. Therefore, just two components are displayed: a Pascal notation of an unknown (investigated) algorithm and a table with the series of its results. To shed more light on the relationship, all intermediate and final values of all variables are displayed. Figure 1 shows the content of the screen for the input $m = 67$ and $n = 9$. The students can change the content of input values—the shaded fields in the second row of the table—by typing values into them. This is the sole activity the students can perform. Our readers are invited to do the same.

As the stage is built inside a spreadsheet, each change results in recalculation of the table. Its new content shows the program execution for the new input pair. As the number of the loop repetitions may differ, the length of the table may change as well. At the end of the recalculation, only the fields "generated by the program" are displayed. All others remain empty. The spreadsheet calculation is hidden; the students can only observe its effects—identical to the effects of the algorithm execution.

Notice that similar simulations are called "tracing" and recommended in introductory programming lessons. In spite of its value, the method is not very popular. First, manual simulations are time-consuming. Secondly, as the calculations are chained, any error in any of intermediate values results in a wrong sequence and, consequently, misleads students.

4 Experiments

The described stage capable of performing calculations is presented to students. They are invited to freely change its input values in order to specify what elementary calculation the algorithm performs. In any boundless environment, it is rather easy to start blundering and get lost. The time volume required by the application of active learning methods is a big concern of their proponents and opponents [22]. To minimize the time losses, the stage must be properly "marked". The marks—supporting learning materials—indicate a proper direction of the research and help the students to clarify the meaning of their partial results. In our case they have a form of questions pointing to specifics of the algorithm behaviour:

The input values m and n must be integers.

1. What values of m prolong the table?
2. What n prolongs the table?
3. What values of m make the table shorter?
4. What n shortens the table?
5. Find a pair of m and n generating a table with exactly four rows.
6. Is there another pair (or pairs) of input values generating four-row table?
7. In your opinion, how many pairs generate four-row tables?
8. Is there any relationship between the final value of k and the number of rows?
9. Find an input pair producing 6 as the final value of k.
10. Are there other pairs of input values generating the final $k = 6$?
11. As the first row is used for the program input, it is always presented. Thus, the minimum table length is one. Some pairs of m and n do not generate any new rows. Find such a pair.
12. What relationship holds between the input values not generating more rows?
13. Find a pair of m and n generating $n = 12$ and $k = 4$ in the last row.
14. Is there more than one solution to Problem 13? If so, how many? If not, why?
15. Find an input pair m and n generating $m = 5$ and $k = 4$ in the last row.
16. Find an input pair m and n generating the triple $m = 5$, $n = 7$ and $k = 4$ in the last row.
17. Is there a pair m and n generating an infinitely long table?
18. What relationships hold between initial and final values of m, n and k?
19. What does the algorithm calculate?
20. What is the consequence of the answer to Problem 16 to the negative integers?

The students are not required to follow any order of the questions and are allowed to exchange their partial observations. For example, the questions 6, 10, 12, 14 and 15 are good candidates for similar "information exchange". Such an approach not only speeds up the road to the results, but also allows students to compare their strategies and to differ between data collection and their evaluation. Often, students distribute their functions and specialize in some of them—an activity that can hardly be applied in other introductory programming courses.

The investigations should finally lead them to the specification of the algorithm through its behavior—here the investigated algorithm is integer division. Among the initial values, m is the dividend and n the divisor. In the final row, k is the quotient and m the remainder.

5 Facilitating New Knowledge

All investigations are followed by discussions. Their aim is to compare and evaluate students' conclusions, allow them to express their opinions and to derive further facts. The reason is that their (sometimes hectic) experimentation seldom gives them room for critical observations and in-depth analysis. Even if some students disclose the "secret" algorithm rather quickly, we encourage them to complete their research as they often uncover other important properties of algorithms. Some conclusions address general concepts shared by all algorithms like:

- The notation of the algorithm is a permanent, unchanging text. At the same time, the execution of the algorithm can virtually be of any length.
- The length of the execution strongly depends on the input values.
- The same input pair always generates the same output table. Different input pairs may generate same results (in this case, the same k and n).

Their next set characterizes relationships between the table content (e.g. the execution itself) and its prescription in Pascal. Here we point students' attention to data patterns generated by the commands:

- By additional questions like: "What sequence of values is stored in the k column and why? What sequence of values is stored in the m column and why? Why has the value of n never changed"? we enhance comprehension of the Pascal commands. The command $m := m - n$ is represented in the first column as an arithmetic progression with its elements differing by n. The command $k := k + 1$ increases the values along the third column. Finally, the value of n in the second column remains unchanged as it is not a subject of any command.
- We also revisit appropriate questions (e.g. Q11 "What pairs of m and n do not generate new rows?"). Here we point to the fact that its answer can be directly derived from a Pascal command. In particular, the loop condition $m \geq n$ prohibits the execution of the loop for $m < n$. Consequently, neither of pairs satisfying the condition produces new rows.
- By envisioning similar character of changes between two neighboring rows in the entire table we pave a road to the notion of "a loop". The changes are identical because they are the results of repeating identical commands.

The last group reflects the relationship between informatics and mathematics, for example:

- The above algorithm is only appropriate for the division of positive integers ($m > 0$, $n > 0$). Other inputs produce incorrect results. Their division algorithm must (and can) be defined in a different way. With the students' cooperation, the "correct" algorithms can be easily specified.
- All executions for $n = 0$ produce infinitely long tables. The discussion here concentrates on the non-existence of the division algorithm (as no value can be divided by zero).

6 Conclusion

Compared to the traditional "command-driven" approach (starting with the description of the programming language commands), our approach can be characterized as "data-driven". Our one goes in an opposite direction: The presentation of results precedes the description of commands that produce them. Our students derive commands from their meaning and way of evaluation.

One can object that similar results can be achieved using other visualization tools e.g. by debuggers. On the other hand, achieving a similar level of comfort, the program must have two separate windows—for the text and for the data. The program is run inside its window; the results are recalculated (and displayed) in the other one. Our "program" is just a picture. This means that it can be instantly replaced by its equivalent in a different programming language—and the rest may remain intact. In doing this, one can stress the importance of the data processing method and its dominance over the notation(s). There are many situations when programmers pay more attention to the meaning of the algorithm than to its notation—the choice of test data is an example. By comparing the alternative notations, the students start understanding similarities and differences between various programming languages. This sort of knowledge is independent on a programming language, its hardware and software implementation. We believe that by stressing these aspects we target the core of programming, the basics of Information Science.

Naturally, it would be naïve to expect that a deep comprehension of these notions will be gained after solving the above problem. Therefore, we have formed and implemented a set of similarly oriented problems. Another similar set oriented to the programming language Karel the Robot is under preparation.

7 Closing Remarks

The animation (input data modifications, instant recalculations as well as the changes of the table length) are the most important components of the success of this and any similar teaching methodology. Therefore, interested readers are encouraged to experiment with the attached copy of our program, and to contact the authors for other, similar ones.

For those who are willing to create similar environments by themselves, this is our "trade secret". The length of the table does not fluctuate. It is permanent and—in the given example—occupies the first three columns of the entire sheet. The first row contains the names of the variables; the second row is reserved for their initial values (exactly as Figure 1 shows). All remaining rows (starting with the third one) contain a series of three intentionally modified formulas:

- Under "standard circumstances", the cell `A3` would contain the formula `=A2-B2`. Instead, it contains `=IF(A2>=B2,A2-B2,0)`. So, its content is evaluated in the expected way when the loop condition holds; otherwise it is replaced by zero. Analogous formulas have been copied to all remaining cells of the first column.

- Similarly, the cell B3 contains `=IF(A2>=B2,B2,0)` instead of "standard" `=B2`.

- Only the third column contains the "obvious calculation" i.e. `=C2+1` in the cell `C3` and its equivalents in the remaining cells.

Due to the above conditions, the evaluation of the cells' content runs in the expected way until the loop condition holds. When the loop condition fails, zeros are assigned into the cells in the column `A` and `B`. For us it indicates that the loop execution has terminated. Since that moment no values should be displayed on the sheet. In our Excel applications, it has been done using conditional formatting (*Format / Conditional Formatting*) and by typing `=AND(A2=0,B2=0)` into *Formula Is.* If the formula holds, the font color is set up to "white"—the color of the background.

There is no secret anymore. After each change of an input value, the formulas in all cells are recalculated. The results of the calculations executed “inside the loop” are displayed. The calculations executed “outside the loop” are also performed, but presenting them “white on white” makes them invisible—the length of the visible section of the table seemingly varies.

References

[1] Black, J.B., Carroll, J.M. and McGuigan, S.M. (1987). What kind of minimal instruction manual is most effective. In P. Tanner and J.M. Carroll (Eds.): *Human factors in computing systems and graphic interface.* Amsterdam, North-Holland.

[2] Black, J.B., Thalheimer, W., Wilder, H., de Soto, D., and Picard, P. (1994). *Constructivist Design of Graphic Computer Simulations.* National Convention of the Association for Educational Communications and Technology. URL: http://www.ilt.columbia.edu./publications/cdgcs.html.

[3] Brusilovskij, P., Calabrese, E., Hvorecký, J., Kournichenko, A., Miller, P. (1997). Mini-languages: A Way to Learn Programming Principles. *Education and Information Technologies* **2**(1).

[4] Cognition and Technology Group at Vanderbilt (1990). Anchored instruction and its relationship to situated cognition. Educational Researcher, 20, 2–10.

[5] Collins, A., Brown, J.S. and Newman, S.E. (1990). Cognitive apprenticeship. In L.B. Resnick (Ed.). Knowing, learning and instruction. Hillsdale, NJ: Erlbaum.

[6] Dávid, A., Eliáš, J., Polášek, I. (2000). *Vybrané numerické metódy a ich programovanie v Exceli.* (Selected numerical methods and their programming in Excel–in Slovak). Vydavateľstvo Ekonóm, Bratislava.

[7] Fox, M.B., Metzelaar, L.C. (1999). *Excel 2000 Essential Basic*, Prentice Hall.

[8] Gick, M.L. and Holyoak, K.J. (1983). Schema induction and analogical transfer. *Cognitive Psychology*, **12**: 306–355.

[9] Hejný M. (1990). *Teória vyučovania matematiky, 1. a 2. diel.* (Theory of teaching mathematics – in Slovak). SPN, Bratislava.

[10] Hvorecký, J. (2001). Spreadsheets to Programming. Wei-Chi Yang, Sung-Chi Chu, Zaven Karian, Gary Fitz-Gerald (editors): *Proceedings of the Sixth Asian Technology Conference in Mathematics*, ATCM 20001, Melbourne.

[11] Hvorecký, J., Trenčanský, I. (1998). Recursive Computations in Spreadsheets. In Wei-Chi Yang, Kizoshi Shrirayanagi, Sung-Chi Chu, Gary Fitz-Gerald (editors): *Proceedings of the Third Asian Technology Conference in Mathematics*, Springer, Tokyo.

[12] Hvorecký, J., Trenčanský, I. (2000). Expanding Spreadsheet Calculations. In Wei-Chi Yang, Sung-Chi Chu, Jen-Chung Chuan (editors): *Proceedings of the Fifth Asian Technology Conference in Mathematics*, Chiang Mai, Thailand.

[13] Jacoby, L.L. (1978). On interpreting the effects of repetition: Solving a problem versus remembering a solution. *Journal of Verbal Learning and Verbal Behavior*, **17**.

[14] Johnson, D. and Johnson, R. (1975). *Learning together and alone.* Englewood Cliffs, NJ: Prentice-Hall.

[15] Kaasbøll, J. J. (1998). Exploring didactic models for programming. *Norsk Informatikk Konferanse NIK'98*, Agder University College, Tapir, Norway.

[16] Kalas, I., Blaho, A. (2002). Exploring visible mathematics with IMAGINE: Building new mathematical cultures with a powerful computational system. *Learning in School, Home and Community.*

[17] Kepencay, M. (2002). *Propedeutika objektovo orientovaného programovani*a (Introduction into Object-oriented Programming–in Slovak), Univerzita Konštantína Filozofa, Nitra.

[18] Kukolíková, D. (2000). *Priestorový mikrosvet na vyučovanie programovania* (A Spacial Microworld for Teaching programming–in Slovak), Univerzita Konštantína Filozofa, Nitra.

[19] Lewis, C., and Olson, G. (1987). Can principles of cognition lower the barriers to programming? *Empirical studies of programmers: second workshop.* Ablex Publishing Corp., Norwood, NJ.

[20] Lovászová, G., and Hvorecký, J. (2001). Spreadsheets and Languages. In Wei-Chi Yang, Sung-Chi Chu, Zaven Karian, Gary Fitz-Gerald (editors): *Proceedings of the Sixth Asian Technology Conference in Mathematics*, ATCM 2001, Melbourne.

[21] Lovászová, G., and Hvorecký J. (2002). When There is More Ways to Get There... In Wei-Chi Yang, Sung-Chi Chu, Zaven Karian, Gary Fitz-Gerald (editors): *Proceedings of the Seventh Asian Technology Conference in Mathematics*, ATCM 20002, Melakka (Malajzia).

[22] Lyons, P. (1992). *Thirty-five lesson formats: A sourcebook of instructional alternatives.* Englewood Cliffs, NJ: Educational Technology Publications.

[23] McIver, L. (2002). Evaluation Languages and Environments for Novice Programmers. In J. Kuljis, L. Baldwin, R. Scoble (Eds.): *Proceedings of 14th workshop of the Psychology in Programming Interest Group*, Brunel University, London.

[24] Nardi, B. A. and Miller, J. R. (1990): *The spreadsheet interface: A basis for end user programming.* Technical Report HPL-90-08, Software Technology Laboratory, HP Laboratories, March 1990.

[25] Spiro, R.J., Feitovich, P.J., Jacobson, M.J. and Coulson, R.L. (1991). Cognitive flexibility, constructivism, and hypertext. *Educational Technology*, 24–33.

[26] TIMSS Results of Slovak Schools (1999). URL: http://www.spu.sanet.sk/projekty/TIMSS/skoly.pdf