Enhancing Comprehension by Using Random Access Memory (RAM) Diagrams in Teaching Programming: Class Experiment

Leonard J. Mselle

School of Informatics
The University of Dodoma
mselel@yahoo.com

Keywords: RAM diagrams, teaching programming, novice programmers, close tracking of code, programming comprehension, programming skills.

Abstract

This paper presents results of an experiment in which Random Access Memory (RAM) diagrams were used to teach novice students C programming. Students were divided into two groups that were differently instructed. The control group was instructed in the traditional way while the experiment group was instructed with the aid of RAM diagrams employed throughout the course. Examination results from the two groups were compared. Statistical analysis was done and the *Z* value was calculated. The results suggest that the use of RAM diagrams improves programming comprehension and programming skills.

1. Introduction

A substantial number of researchers conclude that mastering programming is difficult for majority of students. Dehnadi and Bornat (2006) report that a substantial majority of students fail in every introductory programming course in every UK university. They argue that despite a great deal of research into teaching methods and student responses, the cause are not yet established.

Yousoof et al. (2007) contend that the source of difficult in programming can be attributed to cognitive overload. Cognitive overload happens in programming due to the nature of the subject which is intrinsically over-bearing on the working memory. It happens due to the complexity of the subject itself. They conclude that the problem is made worse by the poor instructional design methodology used in teaching and learning programming. Regarding cognitive load, Ala-Mutka (2003) points out that in programming, students are required to contend simultaneously with a number of issues. These include syntax, semantics, algorithm design, problem solving and paradigm specifics. Marcia (1992), Tudoreanu (2003), Du Bolay et al. (1986), Vainio (2007) are among those who assert that mastering programming is not easy.

1.1 Research on different approaches in teaching programming

Research works on alternative approaches to teaching programming include studies on the use of different methods to conduct lessons. In this direction, alternatives such as improving effectiveness of lectures combined with discussion groups, problem solving approaches, watching examples of running codes, predicting what happens next and learning by doing have been tried. Other alternatives are the use of graphics and graphical metaphors in program visualization. Another focus has been on new concepts such as roles of variables (Kuittinen and Sajaniemi 2003). So far, none of these studies have claimed to have entirely solved the problem.

This research is aimed at introducing and testing the effectiveness of a new tool which combines in one single object; the computer memory (RAM), together with the syntax, semantics, and the variables. This tool is a modification of trace tables which were used to teach programming in early days. Hoc (1989) observes that the lack of "Representation and Processing System" (RPS) closely

related to the computer operations constitutes an obstacle for novice programmers in learning. RAM diagrams are designed to function as RPS.

There is evidence to support the notion that carefully designed tools can aid programming students to develop accurate mental models and hence facilitate their comprehension of programming (Scott et al. 2007). RAM diagrams as a pedagogical tool are designed to address these issues.

To demonstrate and test the effectiveness of RAM diagrams, this paper discusses program visualization in section 2. The concept of trace tables is revisited in section 2.1. RAM diagrams are introduced in section 2.2. They are demonstrated in section 2.3. Advantages of RAM diagrams are discussed in section 2.4. Their ability to foster problem-solving skills is discussed in section 2.5. Experiment about effectiveness of RAM diagrams is discussed in section 3. Results are discussed in section 4 and conclusions are presented in section 5.

2. Program visualization

Program visualization is an approach to teach programming by showing (animating) the code, line by line while vividly reflecting results of its execution. Use of visual representations is not new in programming. Flowcharts have traditionally been used to visualize program structures (Scott, et al. 2005). Napes et al. predict that visualizing the execution of programs and showing a full life cycle of objects would probably help students (Kuittinen et al. 2008).

Research on development, use and effectiveness of animation tools have been covered extensively by Ben-Ari et al. (2001), and Sajaniemi et al. (2003) among others. Online visualization tools such as Jeliot 2000, PlanAni and BlueJ are among the current program animators. Research on program visualization is currently an area of great interest (Ben Bassat 2001), (Scott et al. 2005), (Stutzle and Sajaniemi 2005).

2.1. Trace tables

Trace tables are among the first tools that were employed to teach programming. As far back as 1960s they were regarded as useful tools in teaching programming to beginners. Trace tables were being used for debugging and teaching programming when Pascal and Fortran were the teaching languages (Tailor 1977). Their mechanism is as shown in figure 1.

```
int x = 0;
int i;
while ( i<5{
x = x + i;
i++;
}
```

I	X
1	1
2	3
3	6
4	10

Figure 1- Example of Trace Table

For reasons not yet clear, trace tables do not feature in modern programming books, teaching notes or syllabuses. No apparent reason is given for this abandonment. In this research, a survey carried out on 56 programming books at four universities in Tanzania and Rwanda found that there was no single title that had made reference to trace tables. Trace tables are simple paper and pencil variable-tracing tools. However, they do not explicitly include RAM in their mechanism.

2.2. RAM Diagrams

RAM diagrams, as a tool for teaching programming, are a modification of trace tables. Like trace tables they focus intensely on variables and their values. In addition, RAM diagrams display the code while mimicking RAM in relationship with the change in the behaviour of variables along with the algorithm. RAM diagrams can be used by the instructor to simultaneously reflect the syntax, semantics and the algorithm. This simplifies the task to describe the internal and logical aspects of programming such as variable declaration, data feeding, sequence, bifurcation, iteration, parameter passing and file handling. The first part of a RAM diagram is the header/footer, stating the aspect of programming that is being demonstrated i.e. ***variable declaration, data feeding, selection, iteration, etc***. The second part is the RAM-image, which is represented by an array of cells (rectangles). This gives them ability to mimic the computer RAM in association with variables and their behaviour when the code executes. The third part is the piece of code that is being discussed. This gives them ability to represent the syntax and the algorithm of the problem being solved or described.

Du Boulay et al. (1989) argue that there are two approaches in teaching programming. The first approach is called *black box approach*. Under black box approach, the mechanisms by which the computer operates are hidden from the user. The second approach is *called glass box approach*. In this approach, the user attempts to understand what is going on inside the computer. Each command results in some change in the computer and these changes can be described and understood. Users do not need to become electronic experts. There is an appropriate level that Mayer (1986) refers to as the "transaction level". As a pedagogical tool, RAM diagrams are designed to enable the instructor and the learner to pursue the *glass box approach* in teaching and learning programming. They provide possibility for close tracking with absolute precision.

2.3. Demonstration of RAM diagrams

Consider the following code:

```
/*Program 1*/
 main()
 {
int x;
int y;
x=4;
y=7;
x=x+y;
}
```

Using RAM diagrams, *program 1* can be close-tracked as shown in figure 2.

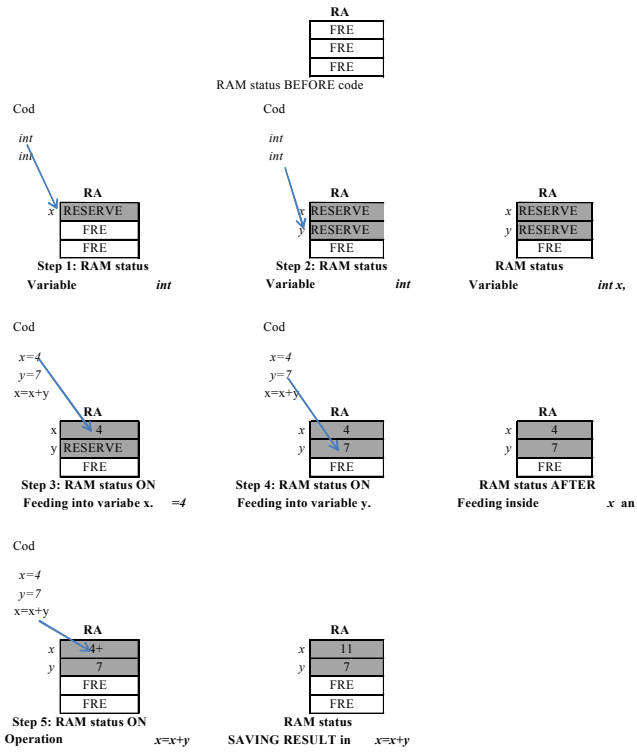

Figure 2 - Example of RAM

2.4. Advantages of RAM diagrams

Du Boulay et al. (1989) offer two important properties for making hidden operations of a language clearer to novice: (1) *simplicity*- there should be "a small number of parts that interact in ways that can be easily understood"; (2) *visibility*- novices should be able to view selected parts and processes" of the model "in action". RAM diagrams have been designed in consideration of the following properties:

1. In order to employ or understand RAM diagrams, students are not required to learn any new concept. For example, to use flow charts, novices are required to understand symbols, their connectivity, and their correspondence with the logic of the code.

2. RAM diagrams portray a direct relationship between code-statement and its effect on the memory (RAM) and the variables. Flow charts and trace tables are devoid of code-RAM relationship.

3. Roles of variables as discussed by Sajaniemi et al. (2003) can explicitly be expressed by RAM diagrams. With RAM diagrams a novice is able to see a gatherer gathering, a stepper stepping, a fixed-value fixed, a follower will be seen following, etc.

4. RAM diagrams are not machine dependent. They are applicable to any programming language. They can be employed within or outside computer environment. This gives them advantage of portability, flexibility and scalability. A programmer does not need to be tied to a specific computer environment (as may be necessary for simulators) to check the precision of the code. With pencil and paper the novice is able to verify the precision of the code segment.

5. RAM diagrams can serve as a code designing and code testing tool. They can be used as code studying tool and debugger. Novice programmers can use RAM diagrams to study other people's programs or their own programs to figure out why an instruction appears, where it appears.

6. By explicitly linking code-writing to variables, RAM diagrams can facilitate schema formation due to their ability to associate machine-memory and code development. According to Ala-Mutka (2003), schema formation is one of the most challenging objectives of training novice programmers. Associating programming practice to computer memory at an early stage can help the novice to fathom the reality that the machine is just a passive recipient of programmers' brain-work. This in turn can enhance ability to design algorithm and schema formation.

7. RAM diagrams can provide an alternative in the way teaching materials are presented. Programming books and notes can incorporate RAM diagrams to provide students with a tool to study and design codes.

2.3. Problem solving and use of RAM diagrams

Johnson-Laird (1959) defined mental models as a way of describing the process which humans go through to solve deductive reasoning problems. His theory included the use of a set of diagrams to describe the various combinations of premises and possible conclusions (Haden and Mann 2003). Numerous studies have concluded that most programming students are overwhelmed by the hurdle to build problem solving skills and proper mental models that will enable them to design and write programs. Scott et al. (2005) contend that for many novice programmers a key weakness lies in their problem solving skills. Many novices engage in program development without possessing an appropriate model of an algorithmic solution. The same views are expressed by Garner and Howell (Stutzle 2005).

Using diagrams to explain and describe phenomena, has been employed in different disciplines to enhance comprehension and building mental models. RAM diagrams, have the ability to reflect the image of variables in the computer memory. Like a map for a navigator, they guide the novice towards the solution in an incremental manner. Using RAM diagrams to visualize the effect of code statements on computer RAM, will enable the programmer to achieve the following:

- Organize ideas in a common theme
- Capture what is going right or wrong in the code
- Understand how the process works
- Understand the way factors (statements and variables) affect one another.

These, in turn, happen to be problem solving steps which, if mastered by novice programmers at the early stage, their abilities to devise algorithms and develop mental models could be largely enhanced.

Pekins et al. (1986) categorize novices programmers into movers, stoppers and tinkerers. Using RAM diagrams, movers can rectify their codes as they move on. Stoppers can use RAM diagrams to chart out new direction. Tinkerers can use them to verify their bearing. For all categories, RAM diagrams constitute an ideal tool for close tracking code (Soloway and Spohrer 1989).

3. Experiment

To test hypothesis that, ***consistent use of RAM diagrams in teaching programming will increase student's ability to devise algorithms and write codes***, a class experiment that included two groups of students, (n=100) was carried out. The groups consisted of first year students who were pursuing introductory programming course at Kigali Institute of Science and Technology (KIST). Both groups

comprised novice students who were being taught Programming in C. The syllabus and teaching hours were equal for both groups.

3.1. Method

The control group, comprising 39 students, was instructed by a lecturer who followed the traditional approach. Lectures were allocated 30 hours while laboratory and tutorials were allocated 45 hours. The experiment group, consisting of 61 students was instructed by a lecturer who employed RAM diagrams. During lectures and tutorials the instructor of experiment group, consistently employed RAM diagrams to explain the code. Equally, during laboratory sessions, students were encouraged to close track their codes using RAM diagrams. At the end of the course both groups attended the same final university examination.

3.2. Subjects

The subjects were undergraduate first-year students studying Introduction to Computer Programming: C Language. The experiment group consisted of students taking food science major. The control group comprised students studying computer science. None of the students had prior exposure to programming. Participants were not made aware of the experiment.

3.3. Materials

The examination consisted of eight questions. Each question carried a total of twenty marks. All questions involved parts of coding and problem solving. The examination was designed to ensure that aspects of sequence, selection, iteration, functions and file handling are well covered among the eight questions from which students were required to select five. The examination and marking scheme were jointly written by a panel of examiners. For the purpose of this experiment, the lecturer for the experiment group decided to exclude himself from setting the examination. However he was present in setting the marking scheme. Scores were distributed to provide the measure of ability to devise algorithms and convert such algorithms in syntactically correct codes. Among questions that featured were: (a) Given the equation; $y=x^2$, write a code to solve it (4 Marks). (b) Write a code that will solve $y=8+2x^2$ (6 Marks). (c) Given the following scores, {60, 45, 34.5, 67, 45, 60}, write a code that will store them in an array and calculate their total. (5 Marks) (d) Using a *while loop*, write a program to store names and addresses of five students in a file (5 Marks). Questions were broken down into at least 4 parts. The total score for any of 5 questions was 100 marks.

3.4. Procedure

The examination duration was three hours. Answer scripts were handed to the invigilator who latter handed them to examinations department. From there, they were collected for grading by the lecturer of control group who was not aware of the experiment. Grading for each question was carried out using a common marking scheme which, for each question was framed based on the following criteria:

(1) Ability to understand the question and evolve a correct algorithm (50% of the marks)

(2) Ability to write a syntactically correct code corresponding to the algorithm (50% of the marks)

After marking, results were handed to the examination department for recording. The researcher collected the results from examinations department for analysis. Assuming that the scores would reflect comprehension of programming and better programming ability, statistical analysis of scores of all, (n=100) students was carried out.

The null hypothesis is therefore stated as Ho: There is no difference in performance between the groups.

The alternative hypothesis is stated as Ha: Performance of the experiment group will be better than that of the control group.

4. Results and discussion

Results of the experiment are summarized in Table 1.

Experiment Group N = 61		Control Group N = 39	
Mean	SD	Mean	SD
64.67	9.58	60.02	11.98

Table 1 - Results obtained from the experiment

The average score for students in the experiment group was 64.67% while that of control group was 60.02%. The standard deviation was 9.58 for the experiment group while that of control group was 11.98.

A two-tailed statistic test at 95% level of confidence was worked out thus $|Z|>1.96$ yielding the value of 2.01. This is a substantial statistical difference which, allows the null hypothesis to be rejected at 0.05 level of significance.

Samurcay (1986) asserts that programming is not only about producing a solution, but also to make explicit the procedure producing the solution. Teaching programming while consistently using RAM diagrams to understand codes, enables the teacher to make explicit the procedure while pursuing the solution (Soloway and Spohrer 19890).

Pekins et al. (1986) posit that close tracking of program is an essential skill in programming. It helps novice to find out bugs. RAM diagrams qualify to be an easy tool for close tracking. RAM diagrams provide the novice with a handy tool to understand programming primitives. Understanding of primitives is the foundation for mastery of more complex issues. As demonstrated, RAM diagrams provide a means for stoppers to reason why the machine is behaving as it is behaving. Tinkerers may use it as a clear guide to proceed (Soloway and Spohrer 1989).

5. Conclusion

As demonstrated in section 2.2 and 2.3, the strength of RAM diagrams as a visual tool, emanates from their simplicity and ability to bundle in one unit the memory (RAM), variables, the code (syntax) and the flow of control of a program. Used effectively, they strengthen the sense of programmer being at the centre of programming as opposed to the feeling that the computer is responsible for anything going wrong about the code. They provide an off-line verification tool.

However, there are questionable issues associated with this experiment. While there is a significant difference in the examination scores of the two groups, this could be explained by at least two other reasons: one, it could be that students in the experimental group were smarter or better able to learn programming. Second, it could be that the teacher of the experimental group was just a better teacher irrespective of the RAM diagrams. To address these issues, more class experiments are being carried out in different settings, to determine the effectiveness of the tool.

6. References

Ala-Mutka, K. (2003) Codewitz, Needs Analysis. [On line]. Available: http://www.cs.tut.fi/~edge/literature_study.pdf.

Ben-Ari, M. and Sajaniemi, J. (2003) Roles of variables from the perspective of computer science educators. [Online]. Available : http://cs.joensuu.fi/pub/Reports/A-2003-6.pdf

Ben Bassat, L. R. et al. (2001) An extended experiment with Jeliot 2000. Proc. First International Program Visualization Workshop, University of Joensuu Press, Pavoo Finland, 131-140.

Kann, C. et al. (1997) Integrating Algorithm Animation into a Learning environment. Computers and Education, 28(4), Elsevier Science Ltd. Oxford, 223-228.
Dehnadi, S. (2006) Testing Programming Aptitude. P. Romero, J. Good, E. Acosta Chaparro & S. Bryant (Eds). Proc. PPIG 18.
Dehnadi, S. and Bornat, R. (2006) The camel has two humps (working title). School of Computing, Middlesex University, UK.
Haden, P. and Mann, S. (2003) The Trouble with Teaching programming. Proc. of the 16th. Annual NACCQ, Palmeston North, New Zealand.
Kuittinen, M. et al. (2008) A study of the development of students' visualizations of program state during an elementary object-oriented programming course. ACM Journal of Educational Resources in Computing, 7(4).
Kuittinen, M. and Sajaniemi, J. (2003) First Results of An Experiment on Using Roles of Variables in Teaching. M. Petre & D. Budgen (Eds) in Proc. Joint Conf. EASE & PPIG, 347-357.
Leslie, J. And Waguespack, Jr., (1989) Visual metaphors for teaching programming concepts. ACM SIGCSE Bulletin, 21(1), 141-145.
Lim, M. and Michael, C. (1992) The case for case studies for programming problems. Communication of the ACM, 35 (3), 120-122.
Sajaniemi, J. and Hu, C. (2005) Teaching programming: Going beyond "objects first". [On line]. Available: http://www.ppig.org/papers/18th-sajaniemi.pdf
Scott, A. et al. (2005) A Step back from Coding – An Online Environment and Pedagogy for Novice Programmers. [On line]. Available: http://www.ics.heacademy.ac.uk/events/jicc11/scott.pdf.
Soloway, J. and Spohrer, C. Studying the Novice Programmer. Laurence Erlbaum Associates: Hillsdale, New Jersey, 1989.
Stutzle, T. and Sajaniemi, J. (2005) An empirical evaluation of visual metaphors in the animation of roles of variables. [On line]. Available: http://inform.nu/Articles/Vol8/v8p087-100stut.pdf.
Tailor, R.T. (1977) Teaching Programming to Beginners. ACM SGCSE Bulletin, 9(1), 1977.
Tudoreanu, M. (2003) Designing Effective Program visualization tools for reducing users cognitive effort. Proceeding of 2003 ACM Symposium on software Visualization, ACM Press, San Diego, California.
Vainio,V. and Sajaniemi, J. (2007) Factors in novice programmers' poor tracing skills. *ITiCSE 2007*, 236-244.
Yousoof, M. et al. (2007) Measuring Cognitive Load - A Solution to Ease Learning of Programming. Proc. Of World Academy of Science Engineering and Technology, 20.