Against the “Hello World”

Leonard J. Mselle
Computer Science
The University of Dodoma
Dodoma, Tanzania

Tabu S. Kondo
Computer Technologies and Applications
The University of Dodoma
Dodoma, Tanzania

ABSTRACT

Computer programming is a “two-way thinking process.” The programmer must think and implant his/her thought in the computer in the form of code. In return, the computer must think like the programmer in the way of output. Compilation is the only initial accurate way of confirming that the programmer and the computer are thinking the same way. In case of novice programmers, the compiler feedback does not suffice the need. In some cases it is a source of confusion and despair. To add to this complexity, the way initial programming is taught and the way programming materials are presented to learners goes contrary to the “two-way thinking.” There is a need for another (mediating) language between the compiler and the novice programmer. In this paper, the traditional practice of introducing programming lessons through programs that display a message such as “Hello World” or any other message is debunked. A new visualization approach through Memory Transfer Language (MTL) is proposed. It is proved that MTL is a language to learn programming whereby students are able to employ hands-on, minds-on and “two-way-thinking” approach to develop coding skills.

General Terms

Programming languages, programming materials, visualization

Keywords

Memory Transfer Language (MTL), program visualization

1. INTRODUCTION

What advocates for greater computing education want are more introductory classes at earlier grades. That way, they say, barriers to entry are lower and more students will start taking the advanced programming classes [1]. There is a move in advanced economies to make programming a basic subject just like arithmetic, reading and writing. Scratch is the language of preference for these early classes due to its simplicity. However, scratch cannot be used to produce industrial applications. There is still a need to explore alternative methods for teaching programming to more students without deviating from the conventional languages such as C, C++ and Java.

Programming is a powerful tool for children to learn learning. Children who engage in programming transfer that kind of learning to other things [2]. Concerning programming, Van Someren [3] looked at novices learning Prolog. He came to the conclusion that those who were successful had a mechanical understanding of the way the language implementation worked.

Ziegler and Crews [4] contend that one problem with teaching programming is the lack of support for learners to experience program execution (i.e. what the computer is thinking). Students often do not view a program as a sequence of steps that must be executed one at a time. Rather, they view programs as a collection of statements that execute when necessary (as opposed to sequentially next). Under this assumption, the exact placement of a statement is irrelevant for the correct working of the program. Thus, students often write programs that contain correct statements; but the order of the statements is incorrect. Since the program execution is not visible, the student often does not realize that there is a misconception.

The way programming books and other teaching materials are presented attest to a general lack of visualization approach. In order for one to be able to program he/she should be able to imagine what is taking place in the computer memory (RAM) for each instruction which is executed [5, 6]. That is, he/she should be able to answer the question “is the computer thinking the way I want it to think?” The learner must strive to understand for each line of code, how the computer is responding in order to become a competent programmer. In other words, programming is to understand what is happening in the RAM as each instruction is executed.

Contrary to the expectation, almost all books and other literature for beginners in programming start with a program that displays a message; specifically “Hello World” or “Hello USA” or something similar. This practice is useless at best and counterproductive at worst. This is so because it is completely devoid of thinking. In addition, starting programming by messaging is devoid of continuity. After displaying a message in the beginning, novices are switched to variables, which dominate the entire programming discourse. This leads to a valid question: “why not start programming with variables?” The perverse approach in teaching programming does not stop at “Hello World.” It continues with the use of verbose approach to explain the logic of programs.

Although there have been attempts to introduce visualization as an alternative approach to teach programming, other methods, broad enough to allow authors of programming books to use visualization in the entire courses are a very recent phenomenon which has not been thoroughly tested [5, 6, 7, 8, 9].

2. THE CASE FOR VISUALIZATION BY MTL

2.1 Visualization in programming

Among the human senses, visual perception is the most important (influential) to cognition. Visualization of non-physical information - financial data, business information, abstract concepts, computer program execution, relationships or specifications is a bit challenging. For a concept to be well and easily understood, visualization, whenever possible, can be employed. In teaching computer programming, the fundamental question has been how to effectively employ visualization for beginners.

Rajala et al [10] contend that program visualization is a research area which studies ways of visually assisting learners in understanding behavior of programs. Visualization of programs can be either dynamic or static. Dynamic program visualization tools visualize execution of programs. They usually show how the execution of programs progresses by highlighting parts of the code under execution and by visualizing changes in variable states.

Various researchers have reported the positive impact of visualization in teaching programming [10]. However, visualization approach is not widely popular among programming instructors. Hundhausen et al [11] state that one of the main reasons visualizations are not widely used is because most instructors refuse to use new methods in teaching. Mselle [8] contends that another reason is that visualization is not yet an integral part of the way programming books are written. He argues that most programming books are written as programming-language manuals. Books and most materials intended for programming beginners do not only suffer from visualization malnutrition; they invariably start out badly with the standard output program as the one depicted in figure 1.

```
#include <stdio.h>
int main()
{
      cout<< "Hello World!\n";
}
```

Fig 1: Hello World Sample Code

A "Hello World" program is a computer program that outputs the message "Hello World" on a display device. Because it is typically one of the simplest programs possible in most programming languages, it is by tradition often used to illustrate to beginners the most basic syntax of a programming language, or to verify that a language or system is operating correctly. While small test programs existed since the development of programmable computers, the tradition of using the phrase "Hello World!" or “Hello my name” as a text message was influenced by an example program in the seminal book “The C Programming Language.” The example program from that book prints "hello, world" (without capital letters or exclamation mark), and was inherited from a 1974 Bell Laboratories internal memorandum by Kernighan [12].

However, introducing programming with a messaging goes contrary to the whole idea of learning programming through visualization. Specifically it does not allow the learner to learn learning. It is devoid of a problematic approach. It does not allow for a successful mechanical understanding of the way the language implementation works. It lacks support for learners to experience program execution. It does not enforce thinking, neither in one way nor two ways. Furthermore, programming logic of sequence, selection, looping, arrays, functions and filing has nothing to do with messaging. The entire logic of program and programming rests on variables. That means, programming is based on RAM. So, the best way to start teaching programming is to start with variables as RAM elements because all programming logic revolves around variable and the RAM.

2.2 Program visualization by MTL

Visualization through MTL is proposed by Mselle [7, 8, 9]. Figure 2 through 5 depict examples of presenting programming concepts through visualization by MTL.

The example in figure 2 is a demonstration of the initial code that is used to teach programming using visualization through MTL as a departure from messaging approach, the “Hello World” style. The MTL approach embraces a problematic situation. It insinuates thinking like a computer. It allows for the learner to mimic the machine RAM while working it. It allows the learners to experience program execution line by line while visualizing a program as a sequence of steps that must be executed one at a time. MTL approach as opposed to messaging approach has a continuity effect. Most of basic programming aspects such as selection, looping, arrays, functions and file handling are explained in the same fashion. Figure 3 depicts an example of visualization of arrays under MTL while figure 4 depicts an example of loop visualization under Pointers, being among the most complex aspects of programming are hard to fathom by novices. It is a daunting and frustrating task for novices to comprehend how addresses can be stored and how they function. Visualization through MTL allows novices to relate data storage with address storage as depicted in figure 5.

As demonstrated by Mselle [5, 6, 7, 8, 9], MTL has successfully been used to write entire programming books in visualization mode. Successful mapping of visualization in the entire programming curricula allows the learners to use the approach even if the instructor does not use it.

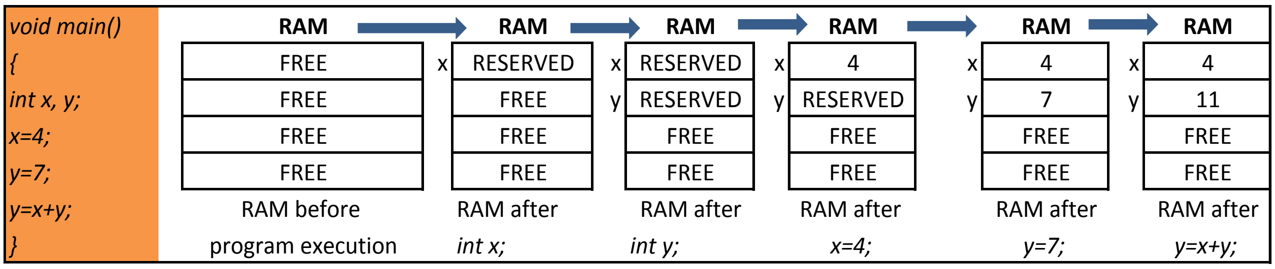
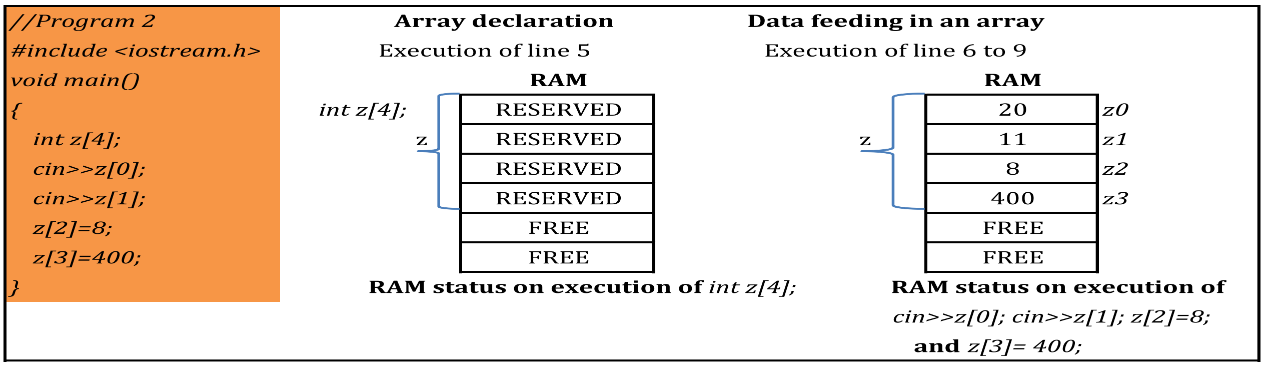
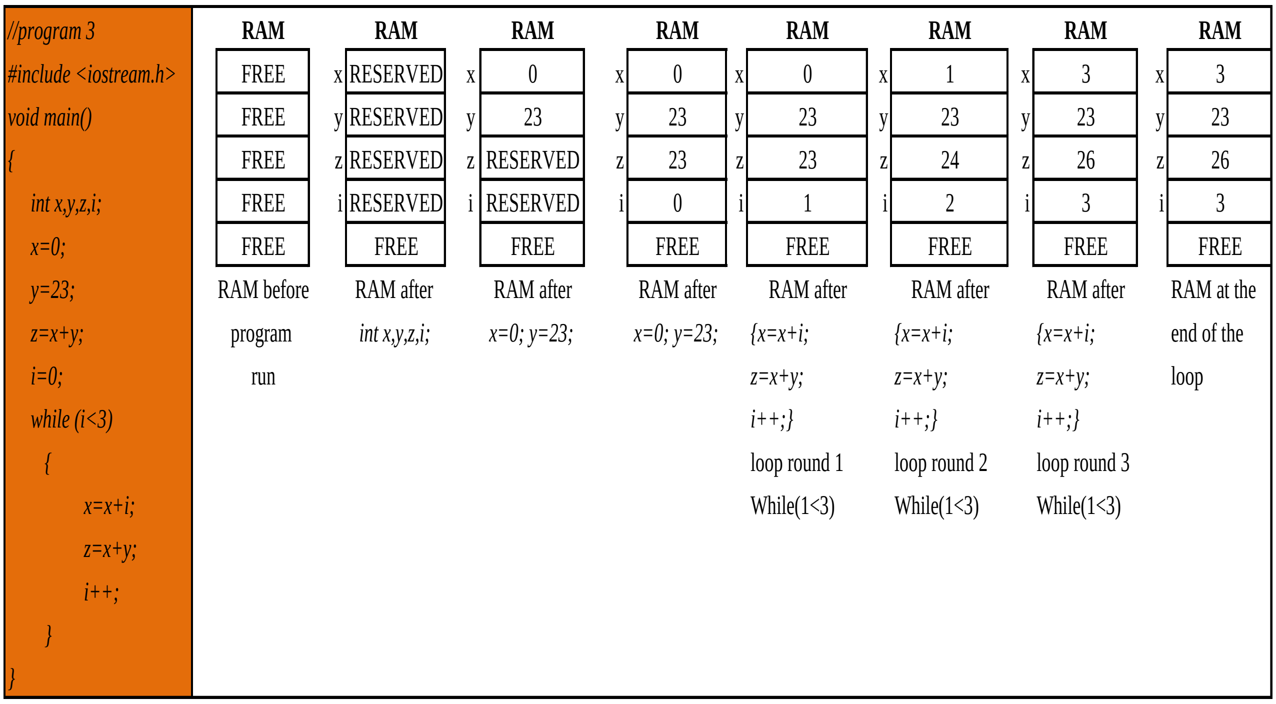


Fig 2: Visualization of Sequence by MTL



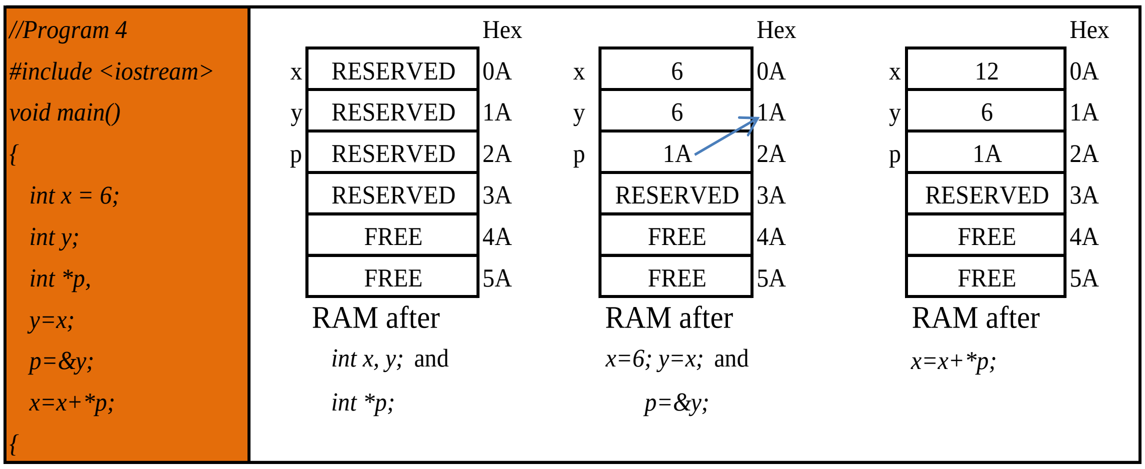
```
//Program 2
#include <iostream.h>
void main()
{
   int z[4];
   cin>>z[0];
   cin>>z[1];
   z[2]=8;
   z[3]=400;
}
```

Fig 3: Visualization of Arrays by MTL



```
//program 3
#include <iostream.h>
void main()
{
   int x,y,z,i;
   x=0;
   y=23;
   z=x+y;
   i=0;
   while (i<3)
    {
         x=x+i;
         z=x+y;
         i++;
    }
}
```

Fig 4: Visualization of Looping by MTL



```
//Program 4
#include <iostream>
void main()
{
  int x = 6;
  int y;
  int *p,
  y=x;
  p=&y;
  x=x+*p;
{
```

Fig 5: Visualization of Pointers by MTL

3. USING MTL WITH PARALLEL LANGUAGES

In other arrangements, during introductory programming, instructors attempt to introduce multiple/parallel languages to emphasize the notion that computers can be made to do the same thing through different languages. This is a useful approach to improve learners' imagination concerning programming and program logic. Since most of conventional approaches rely on verbose explanation of program execution, using a parallel language without visualization approach amounts to more of a burden to the learners. Visualization through MTL provides a possibility of discussing program logic through multiple languages (i.e. C, C++ and Java in tandem). Figure 6 and 7 demonstrate how MTL can be used to represent the employment of parallel languages to teach programming to beginners. The program is meant to perform branching (if else) and division of two numbers.

C Version

```
 1 //Program 1
 2 #include <io.h>
 3
 4 void main()
 5 {
 6    int x;
 7    int y;
 8    double z;
 9    scanf("%d",&x);
10    scanf("%d",&y);
11    if (x>y)
12       z=x/y;
13    else
14       z=y/z;
15    println(z);
16 }
```

C++ Version

```
 1 //Program 1
 2 #include <iostream.h>
 3 using namespace std;
 4 void main()
 5 {
 6    int x;
 7    int y;
 8    double z;
 9    cin>>x;
10    cin>>y;
11    if (x>y)
12       z=x/y;
13    else
14       z=y/z;
15    cout<<z<<endl;
16 }
```

Java Version

```
 1 //Program 1
 2 public class sone {
 3
 4     public static void main(String[] args)
 5 {
 6          int x;
 7          int y;
 8          double z;
 9          x=TextIO.getln();
10          y=TextIO.getln();
11          if (x>y)
12             z=x/y;
13          else
14             z=y/z;
15          System.out.println(z);
16     }
17 }
```

Fig 6: A code in Multiple Languages (Selection)

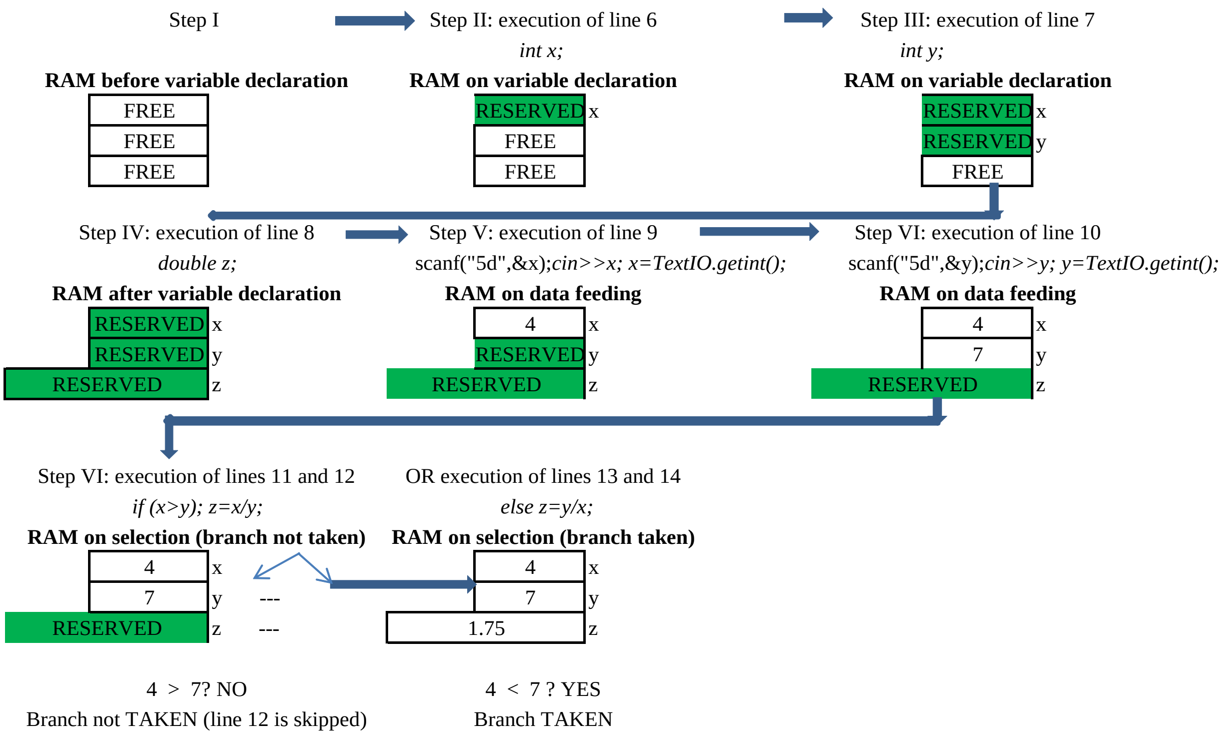

Step I
RAM before variable declaration
FREE
FREE
FREE
Step II: execution of line 6
int x;
RAM on variable declaration
RESERVED x
FREE
FREE
Step III: execution of line 7
int y;
RAM on variable declaration
RESERVED x
RESERVED y
FREE
Step IV: execution of line 8
double z;
RAM after variable declaration
RESERVED x
RESERVED y
RESERVED z
Step V: execution of line 9
scanf("5d",&x);cin>>x; x=TextIO.getint();
RAM on data feeding
4 x
RESERVED y
RESERVED z
Step VI: execution of line 10
scanf("5d",&y);cin>>y; y=TextIO.getint();
RAM on data feeding
4 x
7 y
RESERVED z
Step VI: execution of lines 11 and 12
if (x>y); z=x/y;
RAM on selection (branch not taken)
4 x
7 y ---
RESERVED z ---
OR execution of lines 13 and 14
else z=y/x;
RAM on selection (branch taken)
4 x
7 y
1.75 z
4 > 7? NO
Branch not TAKEN (line 12 is skipped)
4 < 7 ? YES
Branch TAKEN

Fig 7: Single MTL Interpretation of the Programs in Multiple Languages in Figure 6

4. VISUALIZATION Vs HELLO WORLD: THE EXPERIMENT AND RESULTS

Two groups of students (n=63) were subjected to an experiment whereby group one (n=23) was instructed in the traditional “Hello World” mode and another group, (n=40) was instructed using visualization under MTL. These groups were formed by students through voluntary choice after being briefed about the experiment. The first four hours were used for lectures. Latter, both groups were subjected to laboratory works for 4 hours which were used for writing and debugging programs. 2 tutorial hours were added to make a total of 10 learning hours. For the MTL group; lectures, tutorials and laboratory classes were conducted using visualization under MTL approach. Finally a program to perform addition of two numbers in two versions as depicted in figure 8 was used in a quiz to test students understanding. The program in two versions was written on the board and students were told that version 2 was a modification of version 1. The question was whether version 2 would run successfully after being stripped off all messages. Results are summarized in Table 1.

25 out of 63 students (that is 39.68%) replied that version 2 would not run successfully. Of the 25 students, 88% (i.e. 22 students) were from the first group that had been instructed in the traditional “Hello World” mode while 12% (i.e. 3 students) were from the visualization group. It can be concluded that, the use of messages for introducing programming to the novices is inherently inefficient due to the fact that 88% of 23 students who had the wrong point of view were from the group that had been instructed in the traditional approach.

Version I

```
#include<iostrream>
int x, y;
void main()
{
  cout<<”Enter the value of x";
  cin>>x;
  cout<<”Enter the value of y”;
  cin>>y;
}
```

Version II

```
#include<iostrream>
int x, y;
void main()
{
  cin>>x;
  cin>>y;
}
```

Fig 8: Quiz codes

Table 1. Results from the test between MTL and Hello World

Group	N	Incorrect point of view	Percentage
Hello World	23	22	88%
MTL	40	3	12%
Total	63	25	100%

5. CONCLUSION

These results lead to conclusion that early introduction of messaging in teaching programming is deficient and it might have a negative impact in cognition. It confirms the observation by Ziegler and Crews [4] that students often do not view a program as a sequence of steps that must be executed one at a time. Rather, they view programs as a collection of statements that execute when necessary (as opposed to sequentially next). These results also reinforce the finding by Van Someren [3] that students who succeed in programming are those with a mechanical understanding of the way the language implementation works. It has been argued that learning programming is a daunting task [5]. These results show that the standard approach used to teach programming contributes to this difficulty. Animation tools such as Jeliot3, plan Ani, etc. have recently been proposed and results from the use of these tools are encouraging [10]. Good and useful as they may be, animations and visualization using computers do not provide the learner with absolute power to exercise visualization because they are all computer-driven. Another handicap of animation tools is their dependence on digital format which makes it impossible to use them in paper and pencil context. Animations cannot translate programs in multiple languages. MTL approach as demonstrated in this discussion is capable of visually presenting most of programming aspects both digitally and non-digitally. MTL allows the learner to learn to think like the computer from the beginning to the end of introductory courses. MTL allows the learner to visually verify if what she/he sees is what she/he intended the machine to do. It provides a two-way thinking between the programmer and the machine.

Visualization through MTL has not been extended to other programming aspect such as library calling, classes, objects, class overloading and polymorphism. However, MTL approach covers most aspects which are crucial for early understanding of programming. Proper grounding on variables and variable declaration, data feeding, data processing and programming logic covering sequence, selection, loops, functions and file handling can be visualized through MTL as demonstrated in [5, 6, 7, 8, 9].

6. REFERENCES

[1] Benedict, J. H and Du Boulay, B. “Some difficulties of learning to program”. Journal of Educational Computing Research. 2(1). 198. 657–73.

[2] Farrell, M. B. 2013. Offerings limited for advanced courses. Global Staff. [Online]. Available: http://www.michael.farrell@globe.com

[3] Van Someren, M. W. 1990 What’s wrong? Understanding beginners’ problems with prolog. Instructional Science. 19(4/5):256–282.

[4] Ziegler, U. and Crews, T. 1999 “An Integrated Program Development Tool for Teaching and Learning How to Program”. Proceedings of the 30th SIGCSE Symposium. pp. 276-280.

[5] Mselle, L. J. 2014 “Embedding a color Scheme in the Memory Transfer Language (MTL)”. Journal of software engineering and methodology. Volume 4 number 3. pp. 40-47.

[6] Mselle, L. J. 2014 “Using Memory Transfer Language (MTL) as a Tool for Program Dry-running”. International Journal of Computer Applications (0975 – 8887) Volume 85 – No 9. pp. 47-56.

[7] Mselle, L. J. 2010 C++ for Novice Programmers. Lambert Academic Publishing (LAP). Berlin.

[8] Mselle, L. J. 2011 Java for Novice Programmers. Lambert Academic Publishing (LAP). Berlin.

[9] Mselle, L. J. 2011 C for Novice Programmers. Lambert Academic Publishing (LAP). Berlin.

[10] Rajala et al. 2008 “Feedback factor in instruction”, Journal of Education Psychology. Vol. 3. No. 4. 45-67.

[11] Hundhausen, D. C., Sande, W. D. and Sande, C. 2009 Computer Programming for Kids and Other Beginners. Manning Publications Co.

[12] Kernighan, C. B. 1967 Programming in C: A Tutorial – internal Bell Labs memo.

[13] Spohrer, J. and Soloway, E. 1999 “Studying the Novice Programmer”. Lawrence Erlbaum Associates. Hillsdale. New Jersey.