In P. Romero, J. Good, E. Acosta Chaparro & S. Bryant (Eds). Proc. PPIG 18 Pages 244 - 254

# Threshold for the Introduction of Programming: Providing Learners with a Simple Computer Model

Joseph T. Khalife

Lebanese American University
PO Box 36, Byblos, Lebanon
Phone: +961-9-547254, Fax: +961-9-547256
jkhalife@lau.edu.lb

**Abstract.** Computer programming learning/teaching has been an active research area in computer science and engineering. The difficulty level of the teaching/learning process that novices in computer programming report is three-fold, lack of problem solving strategies, misconceptions of code syntax and semantics, and inability to develop an adequate mental model of the machine. This paper examines major difficulties encountered by students taking introductory-level programming courses and it proposes a computer model that sets thresholds for defining basic programming concepts. The study's initial findings suggest that the adoption of the model succeeded significantly in improving students' academic achievement and perception of computer programming.

**Keywords.** Novice Difficulties, Threshold Concept, Computer model.

## 1 Introduction

The learning process in introductory programming courses has captivated the interests of researchers for some time, and considerable work has been done to identify the difficulties encountered by learners. Computer scientists and practitioners have studied novice programmers' difficulties and reported extensively on what they have uncovered. Educational researchers also studied learning in general and introductory-level courses in specific, with their own set of results, of which one important finding is on threshold concepts. The present work aims to identify potential threshold concepts in introductory programming courses and propose solutions to help students surpass thresholds, with the ultimate goal of improving the learning experience for novice programmers.

### 1.1 Threshold Concepts

Understanding threshold concepts by instructors in Higher Education may offer potential help especially to students who acquire formal knowledge of a discipline but who seem unable to apply this knowledge. According to Meyer and Land [1], there exist in many fields threshold concepts, which are akin to a portal opening up a new and previously inaccessible way of thinking about something.

Meyer and Land have also defined threshold concepts as transformative and potentially troublesome. Threshold Concepts are transformative in that they change the way a student looks at things in the discipline. Being conceptually difficult, alien, and/or counter-intuitive, threshold concepts are potentially troublesome for students. Comprehending a threshold concept alleviates difficulties and provides a new way of viewing or understanding a subject, without which the learner cannot progress.

To embed threshold concepts in a first programming course and evaluate whether there is substance to this promise the first step is the identification of such concepts for that course. Moreover, the process of identifying threshold concepts in practice should help to clarify how we should understand threshold concepts in the context of other contributions to the theory of learning [2].

### 1.2 Novice programmers' difficulties

Embarking on a first programming course, learners may encounter difficulties for a variety of reasons, including the inability to develop an adequate mental model of the machine, lack of problem solving strategies, and misconceptions of code syntax and semantics.

Novice programmers may be unable to develop a simple and concrete mental model of the computer internals and how it operates during program execution. As a result, novices will ordinarily tend to incorporate real world, non-programming experiences in their learning of how to program without taking into consideration computer limitations. Common misconception about variables and confusion between assignment and equality are often caused by prior knowledge of algebra.

An ineffective pedagogy in learning programming is a vital factor that influences how a novice programmer will perform in subsequent experiences. A major problem area attributed to pedagogy is the teaching and learning of problem solving skills. This area is often neglected or only briefly considered in early programming instruction. Part of this can be attributed to textbooks, which often dive right into programming without providing support for problem solving as an initial step in program development. When beginners write programs, they frequently generate the source code without any organized thought process. They type their solutions seconds after the initial reading of a problem statement [3] and iteratively proceed by modifying their source code as required until the program generates the correct outputs. Learners of introductory programming are a product of a traditional educational system that

appears to lack adequate emphasis on logical thinking and problem-solving abilities in many of their subjects. General problem-solving strategies should be explicitly acquired along with program development skills [4]. Novices are not familiar with the design and testing of logical structures and heuristics required for solving problems and they have demonstrated lack of ability to perform effective sub-goal decomposition [5], an essential skills in computer programming. It is argued that the task of developing programs may be more difficult for novices than it really should be because it requires solutions to be expressed in ways that are not familiar or natural for beginners.

Fundamental programming concepts are abstract in nature and have no real-word counterparts. Beginners may not have sufficient preparation to grasp such concepts. The syntax of a language also plays an important role in a programmer's ability to use the language. The rigid demands of syntax compared to the inexact and loose nature of the natural, or algorithmic, language result in many students not being able to successfully write programs. Novices lack the ability to correctly debug and comprehend programs. Much of the skills for debugging are learned through the experience of writing and testing programs and since beginners lack adequate program comprehension skills, errors are often inadvertently injected into programs while debugging [6]. Novices have tendencies to understand a program or a portion of a code subjectively. That is, that portion of the code will execute to what it is supposed to do. While debugging, students fail to practice the 'turning their brains off', and to objectively follow the code. As a result, they often correct errors by applying a "patch" to the problem that allows a program to simply compile without understanding what they are doing [7].

### 1.3 Threshold for programming:

The reasons for novices' difficulties described above, as well as many others, stem from a combination of the intricacies of programming languages and their associated paradigms as well as the lack of instructor-related student preparation. Experience shows that many of the students who find it difficult to learn how to program do so when they fall behind. This leads us to the belief that there is perhaps thresholds that students need to pass before they can effectively grasp and utilize the skills that they are being taught.

There are common misconceptions in the way that novice programmers perceive a computer. They have a hard time distinguishing between the "notational machine (the one they learn to control) and its relationship with the physical machine (the computer)" [8]. Computer models may be used to shorten the gap of how a user perceives a programming environment.

We believe that the first threshold students need to pass is to develop a simple but yet concrete mental model of the computer internals and how it operates during program execution. In addition to placing strong emphasis on proper design and modeling prior to coding, we recommend a model that relies on the introduction of a generic instruction set, shields learners from Syntax details, and uses simplified memory

snapshots to visualize the steps of program execution. We believe that such a model could cause a positive shift in students' perception of programming and thus, considered a threshold that students need to pass in order to effectively grasp important programming concepts. In the next section we present a summary of a computer model that can be considered a threshold for the learning of programming.

## 2 A Computer Model as a Threshold for Programming

It is a fact that ample time is needed to discuss computers, their roles, how they work, and how they are programmed. With beginners, understanding computer limitations and acquiring basic problem solving skills, while being carefully introduced to syntax details, are important prerequisites to efficiently learn how to program.

### 2.1 Defining a computer

Many different definitions for a computer are offered. To build a mental model of the machine, novices need to be able to understand and articulate the definition of a computer as a tool that processes data according to a set of instructions, where processing consists of inputting, storing or manipulating, and outputting.

### 2.2 Major Hardware Components

Discussions of major hardware components of a computer should be generic, brief, and limited to prerequisites of understanding how programs are executed. Understanding the purpose of CPU and main memory and the way they cooperate is very important to the study of software development. It should be clear to novices what takes place in main memory during the execution of a set of instructions.

### 2.3 Defining an Appropriate Subset of Generic Instructions

In order to effectively help novices develop an adequate model of the computer, complete coverage of the programming language can be sidestepped, and the focus should be on the collaborative development of a simplified set of generic instructions that could translate easily to any high level language. Learners can augment their knowledge of possible computer instructions as more constructs and instructions are introduced. Because of the intricacies associated with certain paradigms, such as the object-oriented, the initial selection comes from the imperative model and all syntax details are left for subsequent stages, as beginners focus on semantics. Students at this stage should understand that data must be placed in main memory for processing and are introduced to input and output. A simple and best-suited set of computer instructions includes: Declarations, Input, Output and Assignment. Novices can build upon the initial set as more instructions are introduced.

**Declaration.** In memory, data is stored in variables. A variable is a named location in memory. Reserving data space in main memory is accomplished by a declaration statement as follows:

***Type*** <*Variable name*>

Simple example of types includes numbers (Numeric) and words (String). For the computer model we are proposing, there is no need to go into more detail at that stage.

**Input.** One of the many possible input instructions is to read data from Keyboard and store it in a declared Variable:

***KeyboardRead*** <*Variable name*>

During the execution of the *KeyboardRead* instruction, the computer waits for the user to enter some value from the keyboard and press <Return>. The value entered is stored in the variable. In this way, each time the program is run the user gets a chance to type in a different value to the variable and the program also gets the chance to produce a different result.

**Output.** The simplest output device is the computer's screen and this is called standard output. Both values stored in variables and string literals placed between quotes can be outputted. The following is used to denote the output to screen instruction:

***ScreenWrite*** <*Variable name*> *or* <*String literal*>, <*Variable name*> *or* <*String literal*> …

**Assignment.** An assignment statement instructs the computer to evaluate an expression and store the result in a declared variable.

<*Variable name*> **=** <*expression*>

A numeric expression is made up of constants, variables, and numeric operators. The meaning of the equal sign in an assignment statement is very different from its meaning in mathematics. Equality is a totally different programming construct.

The above constructs are considered to be basic, or fundamental, in programming and among the relatively easy task for students to learn and understand their semantics. Next, program composition and the representation of a solution for a problem in terms of these instructions are due.

### 2.4 Problem solving

In the context of developing a computer model, problem solving constitutes an important corner stone. Early introduction of problem-solving tasks such as problem understanding, decomposition, problem modeling, implementation, and testing enhance student's chances of coming to terms with programming.

**Adoption of UML Activity Diagram.** To model a basic computer for instructional purposes, simple UML constructs can be introduced early on and can be built upon as advanced modeling constructs are introduced. The introduction of simplified UML activity diagrams in early instruction forces learners to model solutions to problems in a visual and organized manner. Each of the instructions presented in the previous section is considered an activity or an action. Transition to a next activity is triggered by completion of the previous activity. Directional arrows create links between activities, thus indicating flow between them. Novices can easily understand activity diagrams because of their similarity to flowcharts. At this stage, solutions to simple programming problems could be visually represented in terms of constructs shown in Figure 1.

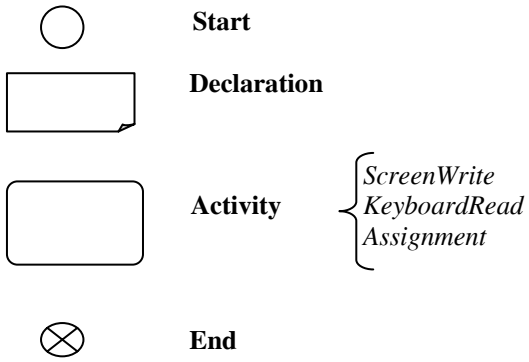


**Figure 1** – Initial Available Constructs

**Program Modeling**. At this early stage, learners have limited knowledge of computer instructions. Yet, this knowledge can be invested in a collaborative manner to guide learners through solving a simple problem, while introducing important parts of the software development cycle.

Consider the problem of finding the sum of three numbers. The first task of a programming problem focuses on the problem understanding. The program goal is to read three numbers, sums them and prints out their total. One possible scenario to accomplish this is to have students collaborate to design and model solutions starting at a high level of abstraction, then decomposing and moving into a level where each activity can be easily translated into its equivalent code. Students' results, taking from an actual introductory tutoring session, are shown in figure 2.

The Third model of figure 2 is very explicit, and it constitutes a decomposition of the previous models. Early in the learning process, novice programmers should be expected to model even the most simplistic aspects of the solution. As students advance however, they will gain more control on deciding at what level of abstraction to stop. Less abstraction will require less coding behind each activity, while more

abstraction will allow the learner greater control over the implementation of their code.

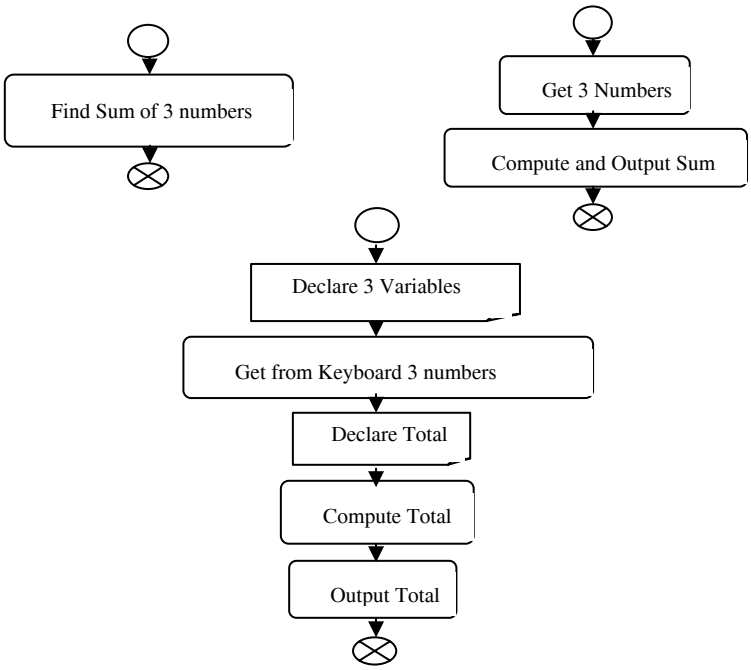


**Figure 2** – Solution models with varying degrees of abstraction

**Coding**. Student next task consists of implementing the solution model into the generic instruction set of section 2.3, drawing attention away from language specific programming syntax. An implementation of the solution model is presented in Figure 3.

***Numeric*** A,B,C
***KeyboardRead*** *A*
***KeyboardRead*** *B*
***KeyboardRead*** *C*
***Numeric*** Total
*Total* = A+B+C
***ScreenWrite****"Sum is Equal to", Total*

**Figure 3** – Generic Implementation

After producing a solution to a computing problem and in order to reinforce a mental model of how the computer works, novices need to acquire an understanding of the dynamic aspect of program execution.

**Program Execution.** A computer program, based on the instruction presented so far, is a sequential process. As program instructions get executed, the program state undergoes various changes. At this stage, program state is best defined as the program variables and their associated values together with the location of the next instruction to be executed. Visualizing the steps of the execution with memory snapshots can effectively assist learners in figuring out what goes on with the program state.

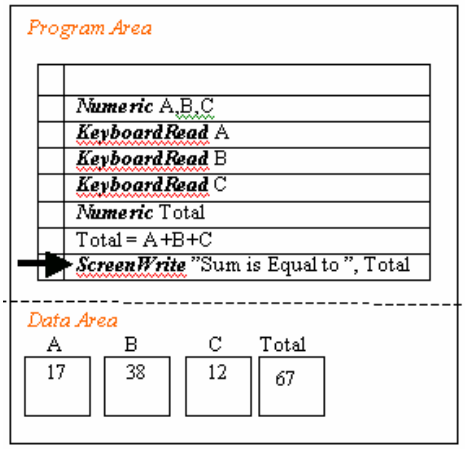


**Figure 4**– Memory Snapshot

The memory snapshot presented in Figure 4 shows the program state after the execution of the assignment statement. It displays, in the program area, the program code and shows explicitly the point of execution. The arrow on the side points to statement to be executed next which happens to be the *ScreenWrite* statement.

Learners can be lead in collaborative manner to notice how the code presented could be improved by using input prompts and thus be introduced to good programming habits. Interactive input, being a major source of confusion to novices, requires clarification if a user enters more input than the program requires, the excess input is stored in a buffer and is used by next input statements requiring interactive input. This can lead to storing non-intended values in input variables. To avoid such problems novices are encouraged to code an appropriate prompt before every interactive input statement.

Showing novices how a computer executes programs completes our model presentation.

## 3 Model Evaluation:

As presented in section 1, a major cause of novice difficulties is their lack of preparation to programming. Accordingly, the 'threshold concepts' computer model proposed earlier will be adopted in classrooms and tested.

### 3.1 Methodology:

This section describes the methodology employed in an attempt to validate the proposed model.

The 'threshold concepts' learning model, adopted in this paper, was introduced at the beginning of Introduction to Object Oriented Programming (csc243), a three credit Java course, required by undergraduate students majoring in Computer Science or Engineering during their first semester at the Lebanese American University. The assessment of the model was done in two stages. The first stage (Fall 2004) involved the introduction of the model on a trial basis to the two sections of csc243. Students' overall performance (letter grade percentages) was compared to students' performance in previous semesters when the model was not used.

The second stage (Fall 2005) assessed model understanding and its correlation to students' performance in their first exam. To determine students' understanding of the model, a short pre-exam1 quiz (Q), consisting of the following three questions on the model, was given and graded.

1- What is a computer and what is programming?
2- Develop a set of possible Instructions that a computer can execute.
3- Based on your instructions set write a program that finds the average of three numbers.

One point was allocated to each question and scores (Q-score) ranged from 0 to 3 (0=no understanding; 1=basic understanding; 2= average understanding; 3= full understanding). Based on their Q scores, students were divided into four categories. Students were then given their first Exam, and an exam1 average (E1-Avg) was computed for students in each of the four categories.

Data was also collected through individual interviews with some csc243 students (Spring 2006), few weeks after being introduced to the model. Students were asked to assess the relevance of the model in helping them grasp programming concepts.

We also interviewed junior and senior students who earlier had some programming difficulties but were later able to overcome their difficulties.

### 3.2 Results:

As shown in figure 8, the adoption of the proposed model in stage I led to a significant improvement in students' performance when compared to a previous course when the model was not used. This is clear in the percentage increases in A's and C's and decreases in D's and F/W's.

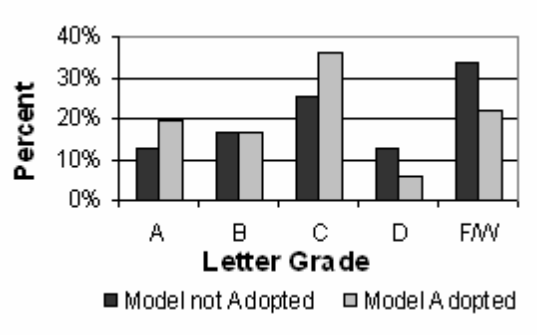


**Figure 8**– Percentage Letter Grades

In stage II, 36 pre-test1 quizzes were completed by students before their first exam. Results of this experiment are presented in Figure 9. The eight students who acquired a full understanding of the model averaged 77.0 on the first exam compared to an average of 28.3 for the four students who scored low on the model (0= no understanding). To determine the amount of correlation between the development of a computer model and students' performance in exams, Pearson's correlation of Q-score and E1-Avg was computed and found to be equal to 0.75. These findings indicate that students who possess a good understanding of the computer model tend to perform better in exams.

| ***Q- Score*** | ***3*** | ***2*** | ***1*** | ***0*** |
|---|---|---|---|---|
| **Number of students** | 8 | 14 | 10 | 4 |
| **E1-Avg** | 77.0 | 60.6 | 50.0 | 28.3 |

Pearson r (Q-score, E1-Avg) = 0.75

**Figure 9**– Comparative Students' Performance

In addition, interviews revealed positive student feedback when asked about the relevance of the model in alleviating programming difficulties. One student shared that he made great progress once able to mentally develop a computer model. Another student, also a tutor in the Learning Center assigned to help weak computer programming students, stated that spending time explaining what takes place in a computer during program execution significantly helped learners overcome many related difficulties.

## 4 Benefits of the Model and Future Work

While relying on flowcharts to introduce programming is not new, the early introduction of a generic instruction set, UML for modeling, and visualization in composition proved advantageous. When the proposed model, simple enough to present no challenge to novice understanding, was used to introduce students to their first programming experience, several of the difficulties encountered by novice programmers were alleviated, simply by helping learners acquire a concrete mental model of what the computer is, and how it operates. Difficulties resulting from rigid demands of syntax were also alleviated by the introduction of a generic set of instruction. In addition, the adoption of a simplified subset of UML in early instruction forced learners to adopt an organized approach to the design and to model solutions to problems before coding.

Future work should be oriented toward precise framing of threshold concepts and the examination of the relationship between threshold concepts and learning difficulties. In other words, this research effort should lead to a systematic method of identification of major threshold concepts in programming as well as the development of teaching material and/or methodologies aimed at helping learners surpass these thresholds.

## References


1. J. H. F. Meyer and R. Land, "Threshold concepts and troublesome knowledge: linkages to ways of thinking and practicing within the disciplines," 10th International Symposium: Improving Student Learning, Brussels, 2002.
2. P. Davies and J. Mangan, “Recognizing Threshold Concepts: an exploration of different approaches”, European Association in Learning and Instruction Conference (EARLI), August 2005, Nicosia, Cyprus.
3. Suchan, W. and Smith, T., “Using Ada 95 as a Tool to Teach Problem Solving to Non-CS Majors, Annual International Conference on Ada”, Proceedings of the Conference on TRI-Ada ’97, Nov. 1997.
4. Pane, J. and Myers, B (August 1996). “Usability Issues in the Design of Novice Programming Systems”, School of Computer Science Technical Report CMU-CS-96-132, Carnegie Mellon University, Pittsburgh, PA.
5. F.P. Deek, J. McHugh, and S.R. Hiltz, “Methodology and Technology for Learning Programming”, Journal of Systems and Information Technology, vol. 4, no. 1, pp. 25-37, June-July 2000.
6. Gugerty, L. and Olson, G. (April 1986). “Debugging by Skilled and Novice Programmers”, Proceedings ACM SIGCHI on Human Factors in Computing Systems, Volume 17, Issue 4, pp. 171-174.
7. J. Bonar and E Soloway, “Preprogramming Knowledge: A Major Source of Misconception in Novice programmers”, in Studying the Novice Programmers, Eds: Laurence Erlbaum Associates, 1989, pp. 325 – 354.
8. Evangeledis, G., Dagdilelis, V., Satratzemi, M. and Efopoulos, V., “X-Compiler: Yet Another Integrated Novice Programming Environment”, Proceedings of IEEE International Conference on Advanced Learning Technologies, pp 166, Aug. 2001.