Towards the Development of Programming Skills for First-Time-Programmers

Ahmed Abukmail[1] and Louise Perkins[1]

Abstract – In this paper we discuss an introductory programming course pedagogy. The target students include both majors and non-majors. It is assumed that the students have had little or no exposure to computer programming but have experience with normal daily use of a computer. To begin the course we started with an introduction to problem solving independent of a specific computer language. After problem-solving skills are explicitly understood, we introduce them to various hardware and software components of a computer. An analogy between a human's problem-solving abilities and the computer's abilities serves to help them understand that a computer will only perform clear and unambiguous instructions. Weekly programming assignments and programming quizzes leverage their newfound software concepts, bringing them to a more explicit understanding of the task of programming. The remainder of the course focused on the beginning concepts of programming including variable declaration, program flow, programming language instructions, input/output operations, assignments and operators, conditional statements, and loops. We also discuss how to develop a program beginning with writing the code in a text editor, compiling, and finally running in an operating system environment. We are proposing this pedagogy to improve non-major instructions as well as to better prepare majors for their first majors-based introduction to computer science course. We support our conclusions and pedagogy using the results of surveys administered to the students at various points during the semester. The results of our survey show significant improvement over monthly intervals.

Keywords: Introduction to Programming, First-Time Programmers, Non-major Programmers, Compilers, Text Editors.

1. INTRODUCTION

When students are first exposed to computer programming, they face a great deal of difficulty. This difficulty stems from the paradigm shift that they face in transitioning from software users to software creators. For this reason we have developed two CS courses, one of which targets new CS majors (we will refer to it as pre-CS1), the other course targets liberal arts students. In this paper we will focus on the pre-CS1 course development, and only briefly discuss the results of the course targeted for the liberal arts.

Many tools have been developed to enhance programming skills for first-time programmers. Karel the Robot [3] has been quite useful in getting students to tell a story via a computer program. JKarelRobot [1] is a newer tool that expands on Karel the Robot by incorporating additional language features and developing new exercises for the students. JKarelRobot is language and platform independent as it supports languages such Pascal, Java, and Lisp. In [4], a list of tools is provided and support for each one of them is discussed. Some of these tools are narrative-based such as Jeroo [6]. We found these narrative-based tools useful.

[1] School of Computing, University of Southern Mississippi, 118 College Dr. #5106, Hattiesburg, MS 39406-0001, {ahmed.abukmail, louise.perkins}@usm.edu

In our pre-CS1 we started by having the students write their first algorithm in the first lecture (a simple program) language-independent. After the first lecture we divided the course into four parts. The first was based on programming with Jeroo [6]. In the second phase we introduce them to the C programming language. Thirdly, we moved to a laboratory setting, where we held the class for two weeks. Finally, we moved back to individual out of class assignments to prepare them for a CS1 setting. Students had weekly programming assignment. We also required students to attend the lectures and take weekly quizzes.

During the classes, we administered surveys at different times during the course. In section 2, we will discuss how students developed their first algorithm on the first day. In section 3, we will discuss Jeroo [6], Section 4 will talk about how we introduced them to C and why we moved them to the laboratory setting. In section 5 we will discuss the part of the course that transitions the students to the CS1 setting. In section 6 we will discuss the results of our survey. In section 7 we will talk about our course targeting liberal arts, and in section 8 we will present our conclusion and future work.

2. DEVELOPING THE FIRST ALGORITHM

As soon as possible, we wanted to show the students that they were capable of writing programs because many students come into their first programming course afraid or nervous. Not only on the first day, but also throughout the course, it was essential to continue to alleviate the students' fear and make them more comfortable with the concepts and technologies utilized in software development. It was also essential to keep pointing out that we were not teaching them how to be proficient in developing algorithms but rather just beginning to learn how to think about developing algorithms.

During the first lecture we posed the question of multiplying two numbers. One of the students answered by multiplying two specific numbers and gave the answer. We explained that these may not be the two numbers needed. A discussion hammered out how a computer would need input just as a human would need input. So, the instructor asked the student to multiply 7 and 5 to which the student actually said "OK". The instructor then posed a question to the class: Why didn't he give me the answer? The student then replied by saying: "You gave me two numbers, and asked me to multiply them, but never asked for the results". At this point a discussion about how a program produces output occurred.

At this point the students were fairly comfortable with the fact that it was actually possible to write computer programs. All you have to do is think about it just like you think about giving instructions yourself. Next we introduced the concept of memory to the students. We asked the students to multiply two 6 digit number in their heads; they could not without writing them on paper. Memory, we explained, is needed to process information: we input into memory and write output from memory. This also allowed us to discuss with the students the various components of a computer as they relate to input, output, processing (CPU), and storage (disk and memory).

The concepts of input and output into and from memory was repeated during the course to make sure the students understood that running a program without the input and without the output is not helpful. Also, they also needed to understand that just like we cannot process information unless it is in memory, the computer is not able to do so either.

3. PROGRAMMING BY STORY TELLING

Repeating the mantra that computers will only do what they're told in the order in which it is told was not enough. It was necessary to prove this point to the students to let the students visually see what the computer does, based on what they're telling it to do. For this we used Jeroo [6].

Jeroo is a software program that uses a GUI that represents a Kangaroo-like creatures that lives on an island. They can only move by hopping in one of the four main compass directions, and they can only turn by 90 degrees angles (left or right). Each Jeroo can pick flowers and place them in its pouch. It can also plant the flowers (assuming that it has them in its pouch). There may be water (blue squares) on or surrounding the island, and there may also be nets (red squares) placed on the island to trap a Jeroo. If a Jeroo hops in the water or in a net, then it can not get out and the program halts. The Jeroo has the capabilities of checking to see if there's water or if there's a net placed in any

of the surrounding cells. A Jeroo can hop a single hop, or multiple hops. It can turn right or left. A Jeroo that has a flower can toss it over a net to disable it. More information about the simulation can be found at (www.jeroo.org).

Jeroo's world can have up to four Jeroos on a 24x24 grid. Using the Jeroo's editor, place nets, water, and flowers on the island to create an initial configuration which the user can manipulate.

The first assignment was to implement an example developed in class. The students had to download the Jeroo software, implement the program and run it. The second assignment was a little more involved as it asked the students to download the map in Figure 1, and use it as the Jeroo's world, and move all the flowers from the corners to the center of the map. This allowed the students to use the if-statement (which they did successfully).

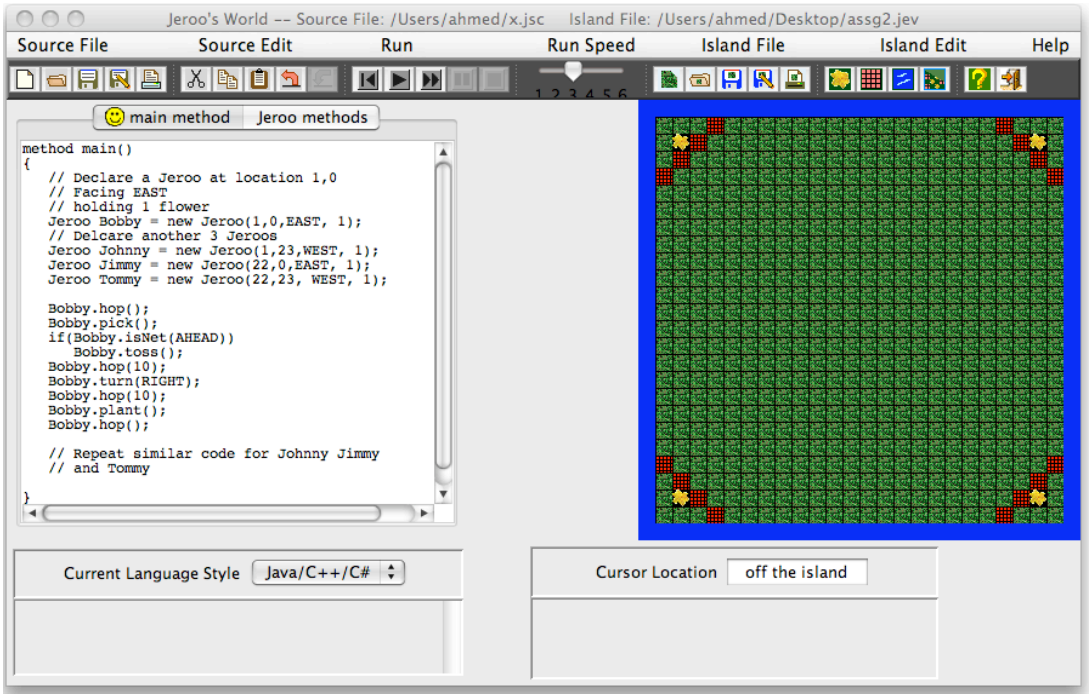

Figure 1 - Jeroo world and second assignment before its execution

In Figure 2 we show the Jeroo world after the execution of code shown in the editor window of the application (we show only the Jeroo Bobby)

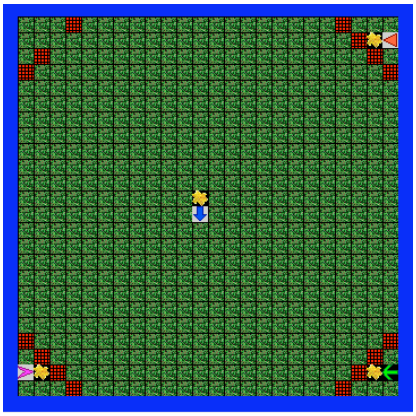

Figure 2 - Jeroo world after the execution. Notice the other three Jeroos were created next to each flower (the adjacent pointer indicating the direction they're facing)

The third Jeroo assignment was the one provided on (www.jeroo.org) - the hurdle race assignment. In this assignment a Jeroo is facing east at the bottom of the map, and there are hurdles (vertical lines consisting of nets) between it and a flower anywhere to the east of it. The Jeroo has to find the flower without running into a net or jumping into water. Figure 3 shows this assignment. This assignment utilizes while loops as well as conditionals. Due to the visual aspect of this programming environment, we believe that the students grasped the concept of the loop quicker than when they encounter them in a normal programming language setting.

The students enjoyed working in this environment. They understood that the Jeroo will only do what you tell it to do and in that order. This made it easy to explain to them how values of variables change during an execution as a result of a series of statements.

4. PROGRAMMING IN C

Our next step was to introduce the C programming language. We start with the syntax of a main program. We went back to the discussion of that first algorithm developed and related how they needed to remember two numbers to multiply them. As a result, they needed to write down two variables. We called this "writing down" declaring variables. Then we showed them how to input these two variables, how to multiply them, and how to assign the results to a third integer, and output the result.

At this point we explained the difference between two C data types (integers & floats). We wrote multiple examples of how to read and write single integers, single floats, multiple integers, multiple floats, and a mixture of both types. The explanation of the '&' in the scanf function was omitted, and the students accepted that the '&' just has to be there when reading data.

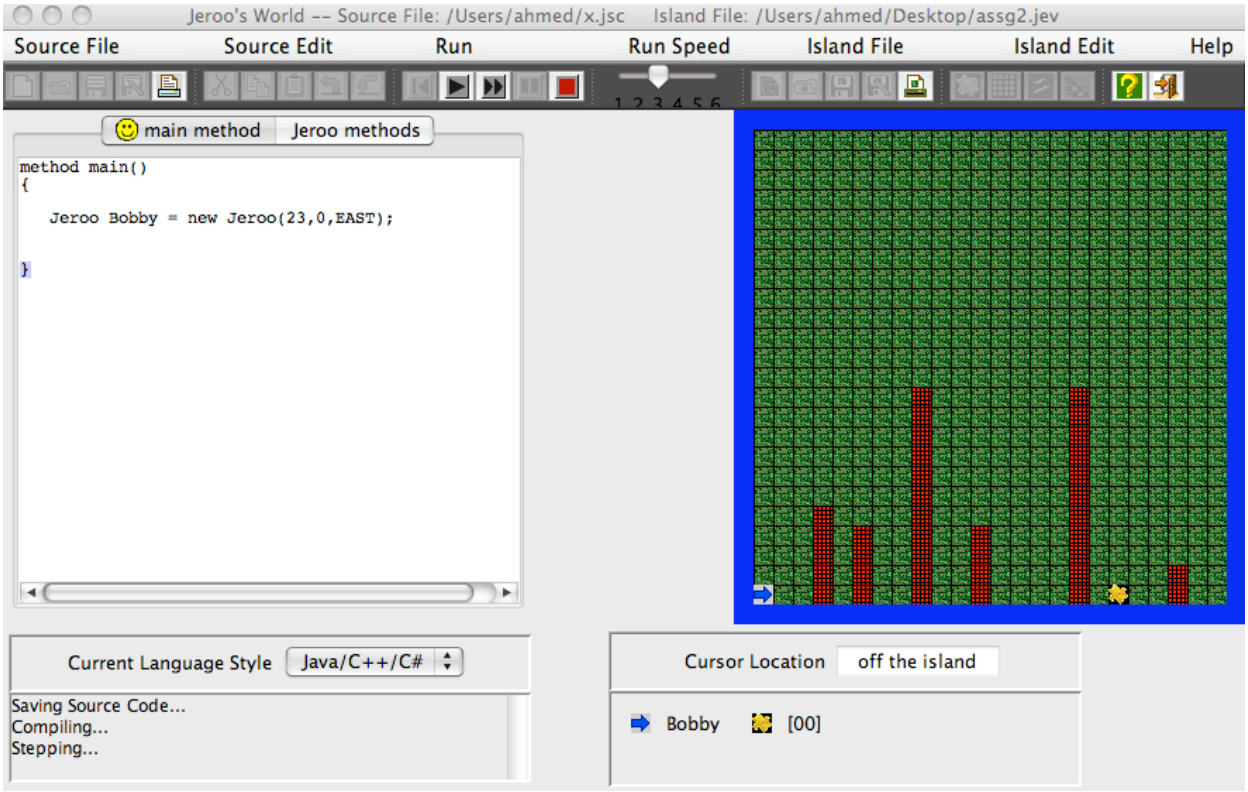

Figure 3 - The Jeroo assignment utilizing loops

4.1 Using the Text Editor and the Compiler

We chose to use a subset of the Linux environment. The emacs editor was chosen due to its language support features. During this part of the course, we stressed the fact that programs are developed using a text editor in a language that is not quite English-like, but is understandable to humans. We showed the students how 'gcc' generates an executable 'a.out' from a single program written and saved using the emacs editor.

We introduced them to the concept of an operating system and the essential role it plays when executing a program. We advised the students to keep two windows open, one for emacs and one for the compilation and execution of the program. The UNIX commands used were, emacs, gcc, ls, cp, more, and exit.

4.2 Individual and Group Assignment

The first assignment given was to read in two integers, add them, multiply them, subtract them, and divide them and output the results. We observed that given the new environment the students had difficulties working on this assignment individually. As a result, we grouped the students in class using groups of 3 students each similar to the work done in [5] where the students would benefit from interacting with their peers. The progress of each student was observed as they work in each group. One of the assignments that we gave was to produce a solution to a quadratic equation. They performed fairly well on this assignment, but still had difficulty implementing it and running it in an operating system environment.

4.3 In-lab Assignments

We started utilizing a computer lab which was equipped with the same environment on which they were asked to develop their programs. The class spent over two weeks in the lab monitoring each student developing code for the assignments. The assignments included calculation of the body mass index (BMI), and the area, circumference, and diameter of a circle given its radius. Once again, the importance of translating input and output into variables and realizing that you must determine I/O before moving on was stressed. The students were extremely satisfied and felt a great deal of comfort in the lab environment while the instructor was present in case problems occur. This gave the students the confidence to work in the lab by themselves as they developed familiarity with the development environment.

5. CONTROL STRUCTURES AND TRANSITION TO CS1 SETTING

At this point the students were familiar with the C programming language environment. They can, with some help, develop simple programs on the environment. Therefore, it was time to introduce them to more complex concepts of C. These concepts have already been introduced in the first part of the course (with Jeroo). We started with the if-statement and explained it in details. Then we gave them an assignment that required the students to read 5 integers and sum them up and produce a message indicating if the sum is positive, negative, or zero.

In order to introduce them to the while loop, we developed two assignments, the first of which is to redo the assignment where they added the five integers, and make it require them to read in as many integers as the user inputs until the user inputs a zero, and produce the same message based on the sum. The second loop assignment was to introduce them to the concept of writing a menu-driven program where they were required to display a message based on the choice entered. The part of the assignment that utilized the loop is that part that makes sure that a correct choice was entered in the correct range (between 1 and 4), the program keeps asking the user for correct input as long and the user inputs incorrect choices.

6. THE RESULTS OF THE SURVEY

Monthly, we administered anonymous student surveys. The three surveys show significant improvement in the students' understanding of the material. Although, the number of students taking the survey was less as the semester progressed, the results were still significantly better when observing the individual responses. Figure 4 shows a chart of the students' progress average throughout the semester. The y-axis represents the Likert scale (1-5), and the x-axis represents the question number (1-16).

We asked the following 16 questions on the survey at the three different times during the fall semester 2009. We were extremely satisfied with the results:

1. I am familiar with what's involved in developing computer programs.
2. I understand the concept of programming.
3. I know MS Windows.
4. I know Linux.
5. I have used a text editor.
6. I know what a variable is.
7. I know the various components involved in building a computer (CPU, memory, hard-drive).
8. I know the function of each component used in a computer (CPU, memory, hard-drive).
9. I know what Input/Output means.
10. I know what a programming language is.
11. I know the function of a compiler.
12. I know how to systematically solve a given problem.
13. I know how to organize and list the steps I used to solve a problem.
14. Rate your math skills (college algebra, trigonometry, calculus).
15. What is your level of comfort in starting to take your first computer programming course.
16. I am motivated and eager to learn how to program.

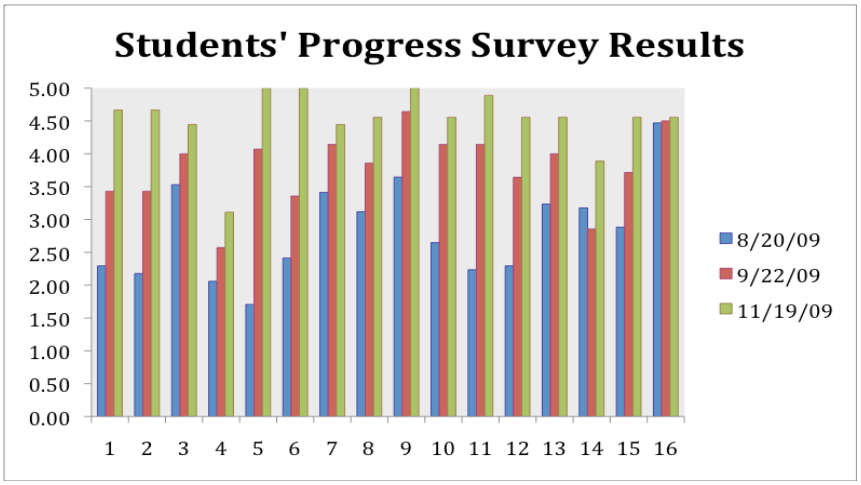

Figure 4 - The results of the survey taken at three different dates during the Fall semester

7. COMPUTER SKILLS FOR THE LIBERAL ARTS

The gap between computer science and liberal arts has never been larger, yet the need to bridge this gap has never been more important. To bridge this gap, liberal arts students need to enhance their computer skills. We offer a programming course for non-majors. Just as operating systems became more user-friendly with the development of a mouse, and point and click GUI's, we anticipate that visual point and click programming, and instant interpreted languages with minimal syntax restrictions may widen the path to programming and allow a larger number of people with various backgrounds to customize their own applications.

Students were given samples of programs from different languages the first day. The language that appeared most intuitive to the students was, surprising, Python. They liked both the absence of brackets, and the easy to use interpreter (previously, the Introduction to Programming Course was taught in C++). We administered a survey in this course as well. However, there is little spread in the data. They indicate that that immediate error detection, simplicity of interface, error location, and rapid turn around are features of Programming that encouraged them to continue.

Some of the assignments that we gave them included:

- Read 10 numbers from a file, convert them to floats, and find the max.
- Use a for loop to move the max number to the top of the list and shift all others down a space
- Add 10 names to go with the numbers and print a "congratulations you've received a scholarship" letter to each name with a number larger than 5.
- Prompt the user with a list of comments and read in their choices as an integer value. Then use if-else to print the appropriate information.
- Add a while loop outside the if-else that loops forever and in the exit if code, use a break to jump out of the loop.
- For the name-gpa program, count the number of students that received a gpa > 3.0 and print their numbers out.

8. Conclusion and Future Work

We believe that our methodologies in both courses were quite successful and we plan to continue developing these courses. We also plan to incorporate the work done in [2] to allow for design-first in the CS1 utilizing UML and interactive tools to learn the concepts of objects. Moreover, we believe that game development is motivational and based on the work in [7], we plan to incorporate game design early on in our CS1, and CS2 courses as well.

References

[1] D. Buck, D. J. Stucki, "JKarelRobot: A Case Study in Supporting Levels of Cognitive Development in the Computer Science Curriculum," In Proceedings of the 32nd SIGCSE Technical Symposium on Computer Sciences Education, Charlotte North Carolina, February 2001, pages 16 – 20.

[2] S. H. Moritz, G. D. Blank, "A Design-First Curriculum for Teaching Java in a CS1 Course," In ACM SGCSE Bulletin, 37(2) 2005, pages 89 – 93.

[3] R. E. Pattis, Karel the Robot: A Gentle Introduction to the Art of Programming, 2nd ed., Wiley, 1995.

[4] K. Powers, S. Cooper, K. J. Goldman, M. Carlisle, "Tools for Teaching Introductory Programming: What Works?," In Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education, Houston, Texas, March 2006, pages 560 – 561.

[5] E. Prather, A.L. Rudolph , G. Brissenden, "Teaching and learning astronomy in the 21st century," Physics Today, 62(10), 2009, pages 41-47.

[6] D. Sanders, B. Dorn, "Classroom Experience with Jeroo," Journal of Computing Sciences in colleges, 8(4), 2003, pages 308 – 316.

[7] M. Zyda, "Computer Science in the Conceptual Age," In Communications of the ACM, 52(12), 2009, pages 66 – 72.

Ahmed Abukmail

Ahmed Abukmail (ahmed.abukmail@usm.edu) is an assistant professor at the University of Southern Mississippi's School of Computing. His research includes computer science education, parallel and distributed simulation, and mobile computing.

Louise Perkins

Louise Perkins (louise.perkins@usm.edu) is a professor at the University of Southern Mississippi's School of Computing on the Gulf Coast campus. Her research interests are natural language processing, data mining, and computational physics.